Contents

DATABASE MANAGEMENT SYSTEM (B.E., IV-SEM.)

PAGE NO.

UNIT - I :

Basic Concepts – Introduction to DBMS, File system vs DBMS, Advantages of database systems, Database System architecture .....(03 to 16)

Data models, Schemas and instances .....(16 to 31)

Data independence, Functions of DBA and designer .....(32 to 39)

Entities and attributes, Entity types, Key attributes, Relationships, Defining, the E-R diagram of database .....(39 to 44)

UNIT - II :

Relational Model – Structure of relational databases, Domains, Relations .....(45 to 51)

Relational algebra – fundamental operators and syntax, relational algebra queries .....(51 to 80)

Entity-Relationship Model – Basic concepts, Design process, constraints, Keys, Design issues, E-R diagrams, weak entity sets, extended E-R features – generalization, specializations and aggregation .....(80 to 118)

UNIT - III :

SQL – Data definition in SQL, update statements and views in SQL, QUEL & QBE – Data storage and definitions, Data retrieval queries and update statements .....(119 to 161)

Query Processing & Query Optimization – Overview, measures of query cost, selection operation, sorting, join, evaluation of expressions, transformation of relational expressions, estimating statistics of expression results, evaluation plans .....(161 to 176)

Case Study of ORACLE and DB2 .....(176 to 178)

UNIT - IV :

Relational Database Design – Functional Dependency – definition, trivial and non-trivial FD, closure of FD set, closure of attributes, irreducible set of FD .....(179 to 186)

Normalization – 1NF, 2NF, 3NF.....(186 to 195)

Decomposition using FD – dependency preservation, lossless join, BCNF, Multi-valued dependence, 4NF, Join dependency and 5NF.....(195 to 216)

UNIT - V :

Introduction of Transaction, Transaction Processing and Recovery ....(217 to 231)

Concurrency Control – Lock management, specialized locking techniques, concurrency control without locking, Protection and Security .....(232 to 258)

Introduction to – Distributed databases, Basic concepts of object oriented database system .....(258 to 280)

UNIT 1

BASIC CONCEPTS – INTRODUCTION TO DBMS, FILE SYSTEM VS DBMS, ADVANTAGES OF DATABASE SYSTEMS, DATABASE SYSTEM ARCHITECTURE

Q.1. What is DBMS ? ***(R.G.P.V., Dec. 2009)***

Ans. A database management system (DBMS) is basically a computerized record-keeping system i.e., it is a computerized system whose overall purpose is to store information and to allow users to retrieve and update that information on demand. DBMS is provide a way to store and retrieve database information that is both conveniently and efficiently.

DBMS are designed to maintain large bodies of information, it is involves both defining structures for storage information and providing modes for manipulation of information furthermore, the DBMS must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.

Q.2. What are the features and objectives of DBMS ? (R.G.P.V., Dec. 2015)

Ans. DBMS has been evolved to overcome the problems and limitations of traditional management, the following are the main objectives of DBMS –

(i) Data Availability – Data availability refers to the fact that the data are made available to wide variety of users in a meaningful format at reasonable cost so that the users can easily access the data.

(ii) Data Integrity – Data integrity refers to the correctness of the data in the database.

(iii) Data Independence – DBMS enables the user to store, update, and retrieve data in an efficient manner. DBMS provides an abstract view of how the data is stored in the database.

(iv) Data Security – Data security refers, to the fact that only authorized users can access the data. Data security can be enforced by passwords. If two separate users are accessing a particular data at the same time, the DBMS must not allow them to make conflicting changes.

Contents

DATABASE MANAGEMENT SYSTEM (B.E., IV-SEM.)

PAGE NO.

UNIT - I :

Basic Concepts – Introduction to DBMS, File system vs DBMS, Advantages of database systems, Database System architecture ..... (03 to 16)

Data models, Schemas and instances ..... (16 to 31)

Data independence, Functions of DBA and designer ..... (32 to 39)

Entities and attributes, Entity types, Key attributes, Relationships, Defining, the E-R diagram of database ..... (39 to 44)

UNIT - II :

Relational Model – Structure of relational databases, Domains, Relations ..... (45 to 51)

Relational algebra – fundamental operators and syntax, relational algebra queries ..... (51 to 80)

Entity-Relationship Model – Basic concepts, Design process, constraints, Keys, Design issues, E-R diagrams, weak entity sets, extended E-R features – generalization, specializations and aggregation ..... (80 to 118)

UNIT - III :

SQL – Data definition in SQL, update statements and views in SQL, QUEL & QBE – Data storage and definitions, Data retrieval queries and update statements ..... (119 to 161)

Query Processing & Query Optimization – Overview, measures of query cost, selection operation, sorting, join, evaluation of expressions, transformation of relational expressions, estimating statistics of expression results, evaluation plans ..... (161 to 176)

Case Study of ORACLE and DB2 ..... (176 to 178)

UNIT - IV :

Relational Database Design – Functional Dependency – definition, trivial and non-trivial FD, closure of FD set, closure of attributes, irreducible set of FD ..... (179 to 186)

Normalization – 1NF, 2NF, 3NF ..... (186 to 195)

Decomposition using FD – dependency preservation, lossless join, BCNF, Multi-valued dependence, 4NF, Join dependency and 5NF ..... (195 to 216)

UNIT - V :

Introduction of Transaction, Transaction Processing and Recovery ..... (217 to 231)

Concurrency Control – Lock management, specialized locking techniques, concurrency control without locking, Protection and Security ..... (232 to 258)

Introduction to – Distributed databases, Basic concepts of object oriented database system ..... (258 to 280)

UNIT 1

BASIC CONCEPTS – INTRODUCTION TO DBMS, FILE SYSTEM VS DBMS, ADVANTAGES OF DATABASE SYSTEMS, DATABASE SYSTEM ARCHITECTURE

Q.1. What is DBMS ? ***(R.G.P.V., Dec. 2009)***

Ans. A database management system (DBMS) is basically a computerized record-keeping system i.e., it is a computerized system whose overall purpose is to store information and to allow users to retrieve and update that information on demand. DBMS is provide a way to store and retrieve database information that is both conveniently and efficiently.

DBMS are designed to maintain large bodies of information, it is involves both defining structures for storage information and providing modes for manipulation of information furthermore, the DBMS must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.

Q.2. What are the features and objectives of DBMS ? (R.G.P.V., Dec. 2015)

Ans. DBMS has been evolved to overcome the problems and limitations of traditional management, the following are the main objectives of DBMS –

(i) Data Availability – Data availability refers to the fact that the data are made available to wide variety of users in a meaningful format at reasonable cost so that the users can easily access the data.

(ii) Data Integrity – Data integrity refers to the correctness of the data in the database.

(iii) Data Independence – DBMS enables the user to store, update, and retrieve data in an efficient manner. DBMS provides an abstract view of how the data is stored in the database.

(iv) Data Security – Data security refers, to the fact that only authorized users can access the data. Data security can be enforced by passwords. If two separate users are accessing a particular data at the same time, the DBMS must not allow them to make conflicting changes.

A typical DBMS has the following features –

(i) The query language facility helps non-programming persons to access the database for information as needed without the help of any programmer.

(ii) It provides a way to structure data as records, tables or objects.

(iii) It provides multiuser access to data along with security features that prevent some users from viewing and/or changing certain types of information.

(iv) It provides data integrity features that prevent more than one user from accessing and changing the same information simultaneously.

Q.3. What are the functions of DBMS ?

Ans. The functions of DBMS are as follows –

(i) Data Definition – The DBMS must be able to accept data definitions in source form and convert them to appropriate object form. In other words, the DBMS must include DDL processor or DDL compiler components for each of the various data definition languages (DDLs).

(ii) Data Manipulation – The DBMS must be able to handle requests to retrieve, update, or delete existing data in the database or to add new data to the database. In other words, the DBMS must include a DML processor or DML compiler component to deal with the data manipulation language (DML).

(iii) Optimization and Execution – DML requests, planned or unplanned, must be processed by the optimizer component whose purpose is to determine an efficient way of implementing the request. The optimized requests are then executed under the control of the run-time manager.

(iv) Data Security and Integrity – The DBMS must monitor user requests and rejects any attempts to violate the security constraints defined by the DBA. These tasks can be carried out at compile time or run-time or some mixture of the two.

(v) Data Recovery and Concurrency – The DBMS, or some other related software component, called the transaction manager or transaction processing monitor must enforce certain recovery and concurrency controls.

(vi) Data Dictionary – The DBMS must provide a data dictionary function. The data dictionary is a database which contains "data about the data" (called metadata or descriptor) – that is, definitions of other objects in the system, instead of "raw data". In particular, all of the various schemas and mappings and all of the various security and integrity constraints will be stored, in both source and object form, in the dictionary.

(vii) Performance – The DBMS should perform all of the tasks identified above as efficiently as possible.

Q.4. Write down six major steps that you would take in setting up a database for particular enterprise. ***(R.G.P.V., Dec. 2009)***

Ans. The six major steps of the database design are as follows –

(i) Requirement Collection and Analysis – Before we can effectively design a database we must know and analyze the expectations of the users and the intended uses of the database in as much detail as possible. This process is called ***requirement collection and analysis.*** To specify the requirements, we must first identify the other parts of the information system that will interact with the database system. These include new and existing users and applications, whose requirements are then collected and analyzed.

(ii) Conceptual Database Design – The goal of this phase is to produce a conceptual schema for the database that is independent of a specific DBMS. We often use a high level data model such as the ER or EER model during this phase. In addition we specify as many of the known database applications or transactions as possible, using a notation that is independent of any specific DBMS. Often, the DBMS choice is already made for the organization. The intent of conceptual design is still to keep it as free as possible from implementation considerations.

(iii) Choice of a DBMS – The choice of a DBMS is governed by a number of factors – some technical, others economic, and still others concerned with the politics of the organization. The technical factors are concerned with the suitability of the DBMS for the task of hand. Issue to consider here are the type of DBMS, the storage structures and access paths that the DBMS supports, the user and programmer interfaces available, the types of high-level query languages, the availability of development tools, ability to interface with other DBMSs through standard interfaces, architectural options related to client-server operation and so on. Nontechnical factors include the financial status and the support organization of the vendor.

(iv) Data Model Mapping (Logical Database Design) – During this phase which is also ***logical database design,*** we ***map*** (or ***transform***) the conceptual schema from the high level data model used in phase 2 into the data model of the chosen DBMS. We can start this phase after choosing a specific type of DBMS – for example, if we decide to use some relational DBMS but have not yet decided on which particular on. We call the latter ***system independent*** (but data ***model-dependent***) logical design. In terms of the three level DBMS architecture, the result of this phase is a ***conceptual schema*** in the chosen data model. In addition, the design of ***external schemas*** (views) for specific applications is often done during this phase.

(v) Physical Database Design – During this phase, we design the specifications for the stored database in terms of physical storage structures.

record placement, and indexes. This corresponds to designing the ***internal schema*** in the terminology of the three-level DBMS architecture.

(vi) Database System Implementation and Tuning – During this phase, the database and application programs are implemented, tested, and eventually deployed for service. Various transactions and applications are tested individually and then in conjunction with each other. This typically reveals opportunities for physical design changes, data indexing, recognization, and different placement of data–an activity reffered to as ***database tuning.*** Tuning is an ongoing activity–a part of system maintenance that continues for the cycle of a database as long as the database and applications keep evolving and performance problems are detected.

Q.5. List any eight applications of DBMS. ***(R.G.P.V., Dec. 2015)***

Ans. Some commonly used applications of DBMS are –

(i) Banking (ii) Airlines
(iii) Universities (iv) Telecommunication
(v) Manufacturing (vi) Human resources
(vii) Sales (viii) Finance.

Q.6. Define the term data redundancy and inconsistency. ***(R.G.P.V., Dec. 2011)***

Ans. In data redundancy, it is leads to higher storage and access cost. Because several programmer create the different files and application programs over a long period, and the various files consist of different formats and program can be written in many programming languages.

Furthermore the same information can be duplicated in several files. In this redundancy it is defining and storing data causes wasted storage space and redundant efforts to maintain common data up-to-date.

In the data inconsistency, that is, the various copies of the same data may no longer agree.

Q.7. Compare file management system with database management system. ***(R.G.P.V., Dec. 2010)***

Or

Discuss the main characteristics of the database approach and how it differs from traditional file system. ***(R.G.P.V., June 2011)***

Or

What are the major differences between a file processing system and a DBMS ? ***(R.G.P.V., Dec. 2011)***

Or

Compare file system with database system. ***(R.G.P.V., Dec. 2012)***

Or

List four significant differences between a file processing system and a DBMS. ***(R.G.P.V., Dec. 2009, June 2015)***

Or

Explain disadvantages of conventional file based system compared to database management system. ***(R.G.P.V., May 2018)***

Ans. There are a number of characteristics to distinguish the database approach from the traditional approach of programming with files. In traditional file processing, each user defines and implements the files needed for a specific application as part of programming the application. For example, one user, the ***grade reporting office***, may keep a file on students and their grades. Programs to print a student transcript and to enter new grades into the file are implemented. A second user, the accounting office, may keep track of student's fees and their payments. Although both users are interested in data about students, each user has to maintain separate files and programs to manipulate these files because each user requires some data not available from the other user's files. This redundancy in defining and storing data causes wasted storage space and redundant efforts to maintain common data up-to-date. In the database approach, a single repository of data is maintained that is defined once and then is accessed by various users.

The main characteristics of the database approach versus the file-processing approach are as follows –

(i) Self-describing Nature of a Database System – The database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the system, ***catalog***, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is ***meta-data*** and it describes the structure of the primary database.

In traditional file processing, data definition is the part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs. Whereas file-processing software can access only specific databases, DBMS software can access diverse databases by extracting the data definitions from the catalog and then using these definitions.

(ii) Insulation between Programs and Data, and Data Abstraction– In traditional file processing, the structure of data files is embedded in the access programs, so any changes to the structure of a file may require changing all programs that access this file. By contrast, DBMS access programs do not require such changes. The structure of data files is stored in the DBMS catalog

separately from the access programs. This property is called ***program-data independence***.

In object-oriented and object-relational databases, users can define operations on data as part of the database definitions. An operation is specified in two parts. The interface (or signature) of an operation includes the operation name and the data types of its arguments. The implementation (or method) of the operation is specified separately and can be changed without affecting the interface. User application programs operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This is known as ***program-operation independence***. The characteristic that allows program-data independence and program-operation independence is called ***data abstraction***.

(iii) Multiple Views of the Data – A database has many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Some users do not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS having variety of applications must provide facilities for defining multiple views.

(iv) Sharing of Data and Multiuser Transaction Processing – A multiuser DBMS must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

Q.8. What are the advantages of DBMS over traditional file system ? ***(R.G.P.V., June 2007)***

Or

What are the main advantages of DBMS ? ***(R.G.P.V., June 2013)***

Or

What are the advantages of using a DBMS ? ***(R.G.P.V., Dec. 2015)***

Or

Write any five advantages of DBMS. ***(R.G.P.V., June 2016)***

Ans. The advantages of DBMS are as follows –

(i) Reduction of Redundancies – Centralized control of data by the DBA avoids unnecessary duplication of data and reduces the total amount of data storage required. It also eliminates the extra processing necessary to trace the required data in a large mass of data. Another advantage of avoiding duplication is the elimination of the inconsistencies present in redundant data files.

(ii) Shared Data – A database allows the sharing of data under its control by any number of application programs or users.

(iii) Integrity – Centralized control ensures that adequate checks are incorporated in the DBMS to provide data integrity. Data integrity means that the data contained in the database is accurate and consistent.

(iv) Security – Data is of vital importance to an organization and may be confidential. This confidential data must not be accessed by unauthorised persons. The DBA ensures that proper access procedures are followed, including proper authentication schemes for access to the DBMS and additional checks before permitting access to sensitive data. Different levels of security can be implemented for various types of data and operations. The enforcement of security can be datavalue dependent as well as data-type dependent.

(v) Conflict Resolution – The DBA resolves the conflicting requirements of various users and applications. In essence, the DBA chooses the best file structure and access method to get optimal performance for the response-critical applications, while permitting less critical applications to continue to use the database, although with a relatively slower response.

(vi) Data Independence – There are two types of data independence – physical data independence and logical data independence. Physical data independence allows changes in the physical storage devices or organization of the files without requiring changes in the conceptual view or any of the external views and hence in the application programs using the database. Thus, the files may migrate from one type of physical media to another or the file structure may change without any need for changes in the application programs. Logical data independence implies that application programs need not to be changed if fields are added or deleted to and from an existing record. Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemas. Data independence is advantageous in the database environment since it allows for changes at one level of the database without affecting other levels.

Q.9. What is the role of a DBMS and what are its advantages ? ***(R.G.P.V., Dec. 2011)***

Or

What do you mean by database management system ? Explain the various advantages of using a database management system. (R.G.P.V., June 2017)

Ans. Refer to Q.1 and Q.8.

Q.10. With the help of block diagram explain the architecture of a DBMS.

(R.G.P.V., June 2014)

Or

Explain the component modules of a DBMS and their interactions with the architecture. ***(R.G.P.V., Dec. 2013)***

Or

Explain DBMS architecture. ***(R.G.P.V., June 2007, 2010)***

Or

Write short note on database management system architecture.

(R.G.P.V., June 2004, Nov./Dec. 2007)

Or

Explain the architecture of DBMS. ***(R.G.P.V., Dec. 2012)***

Or

Discuss the various components of database system.

(R.G.P.V., Dec. 2016)

Ans. A database system is partitioned into module that deal with each of the responsibilities of the overall system. The functional components of a database system can be divided into the storage manager and the query processor components. Fig. 1.1 shows the structure of DBMS.

In fig. 1.1, the functional components of DBMS have query components and storage components as follows –

The Query Processor – The query processor components include –

(i) DDL Interpreter – Its function is to execute the low-level statements and records them in a set of tables that having metadata.

(ii) DML Compiler – Its function is to convert DML statements in source form of query language into necessary object form (i.e., low-level instructions) that query evaluation engine understands.

(iii) DML Precompiler – Its function is to convert DML statements embedded in application program to normal procedure calls in host language. To generate appropriate code, the precompiler must interact with DML compiler.

(iv) Query Evaluation Engine – Its function is to execute the low-level instruction generated by DML compiler.

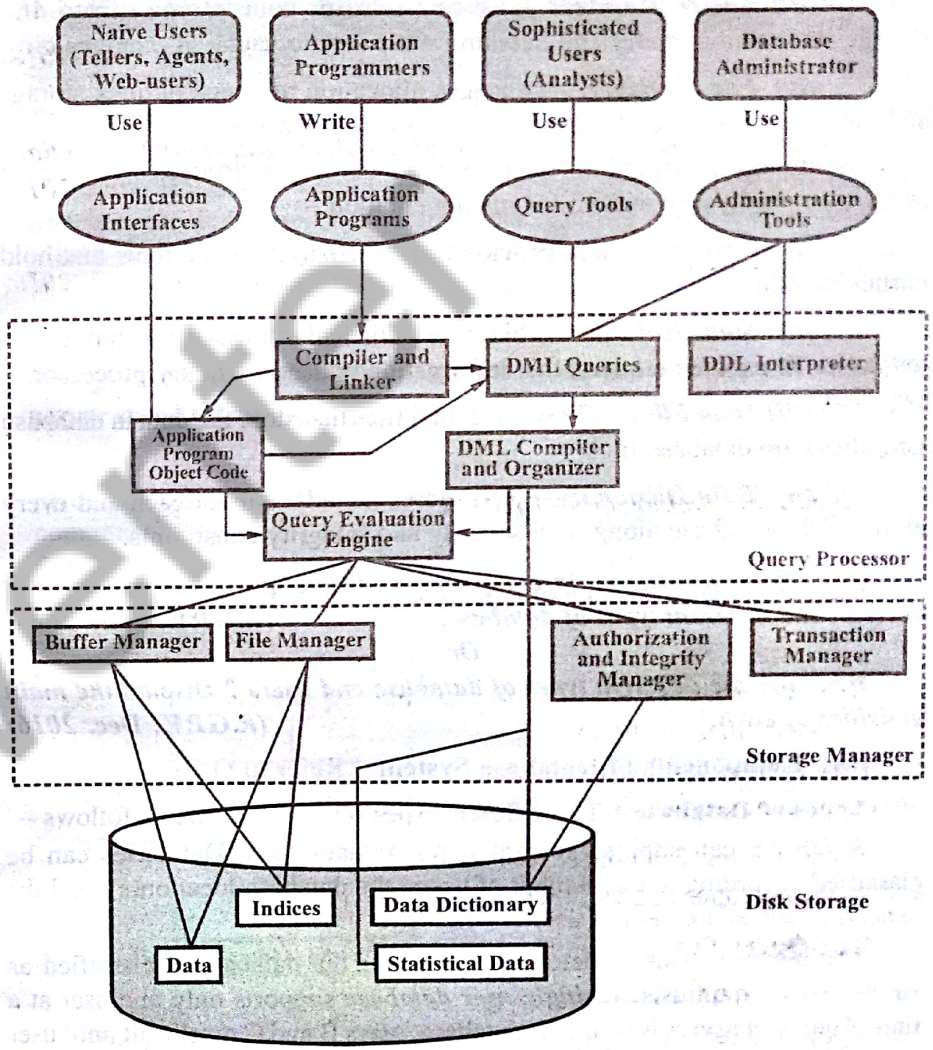


Fig. 1.1 System Structure

Storage Manager – A storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

The storage manager components include –

(i) Authorization and Integrity Manager – Tests for the satisfaction at integrity constraints and checks the authority of users to access data.

(ii) Transaction Manager – Ensures that the database remains in consistent state despite system failure and concurrent transaction executions proceed without conflicting.

MSInvertis

(iii) Buffer Manager – Responsible for data fetching from disk storage into main memory and deciding what data to cache in memory.

(iv) File Manager – Manages allocation of space on disk storage and data structures used to represent information that stored on disk.

The storage manager implements several other data structures are needed as the part of physical system for implementation –

(i) Indices – These provide fast access to the data items that hold particular values.

(ii) Statistical Data – This store statistical information about data in database. To execute a query, this information is used by query processor.

(iii) Data Files – These are actual files that store the data in database i.e., these are database files.

(iv) Data Dictionary – This stores metadata about each and every entity of the database along with security and integrity constraints.

Q.11. What are the main components of database system ? List and describe the different types of database. ***(R.G.P.V., June 2009)***

Or

What are the different types of database end users ? Discuss the main activities of each. ***(R.G.P.V., Dec. 2016)***

Ans. **Components of Database System** – Refer to Q.10.

Types of Database – The different types of database are as follows –

A DBMS can support several types of databases. Databases can be classified according to the number of users, the database location(s), and the expected type and extent of use.

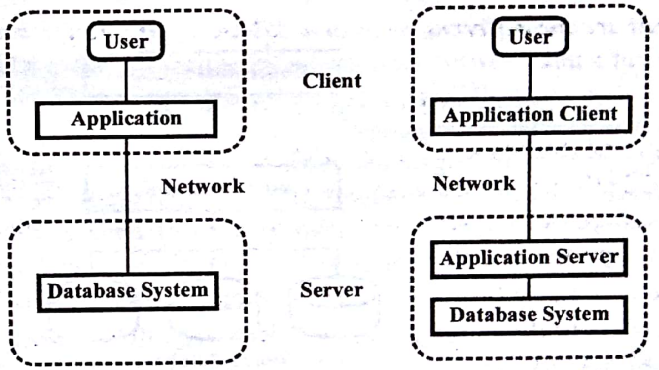
The number of users determines whether the database is classified as single-user or multiuser. A ***single-user database*** supports only one user at a time. That is, if user A is using the database, users B and C must wait until user A is done. A single-user database that runs on a personal computer is called a ***desktop database.*** In contrast, a ***multiuser database*** supports multiple users at the same time. When the multiuser database supports a relatively small number of users (usually fewer than 50) or a specific department within an organization, it is called as a ***workgroup database.*** When the database is used by the entire organization and supports many users (usually more than 50) across many departments, the database is called as an ***enterprise*** database.

Location might also be used to classify the database. For example, a database that supports data located at a single site is called a ***centralized database.*** A database that supports data distributed across several different sites is called a ***distributed database.***

The most popular way of classifying databases today, however, is based on how they will be used and on the time sensitivity of the information gathered from them. For example, transactions like product or service sales, payments, and supply purchases reflect critical day-to-day operations. Such transactions must be recorded accurately and immediately. A database that is designed primarily to support a company's day-to-day operations is classified as an ***operational database.*** It is sometimes referred to as a ***transactional*** or ***production database.*** In contrast, a datawarehouse focuses primarily on storing data used to generate information required to make tactical or strategic decisions.

Q.12. Explain the difference between the two-tier and three-tier client/server architecture. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. Database applications are partitioned into two or three parts as shown in fig. 1.2. In a ***two-tier architecture***, the application is partitioned into a component that resides at the client machine, which invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.



(a) Two-tier Architecture ***(b) Three-tier Architecture***

Fig. 1.2 Two-tier and Three-tier Architectures

In contrast, in a three-tier architecture, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, through a forms interface. The application server in turn communicates with a database system to access data. The ***business logic*** of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more suitable for large applications, and for applications that run on the World Wide Web.

Q.13. Describe the three-schema architecture. Why do we need mapping between schema levels ? How do different schema definition languages support this architecture ? ***(R.G.P.V., June 2011)***

Ans. **Three-schema Architecture** – Fig. 1.3 shows the three-schema architecture. Its goal is to separate the user applications and the physical database. The architecture is divided into three levels – the external level, the conceptual level and the internal level. The view at each of these levels is described by a schema. A schema is an outline or a plan that describes the records and relationships existing in the view.

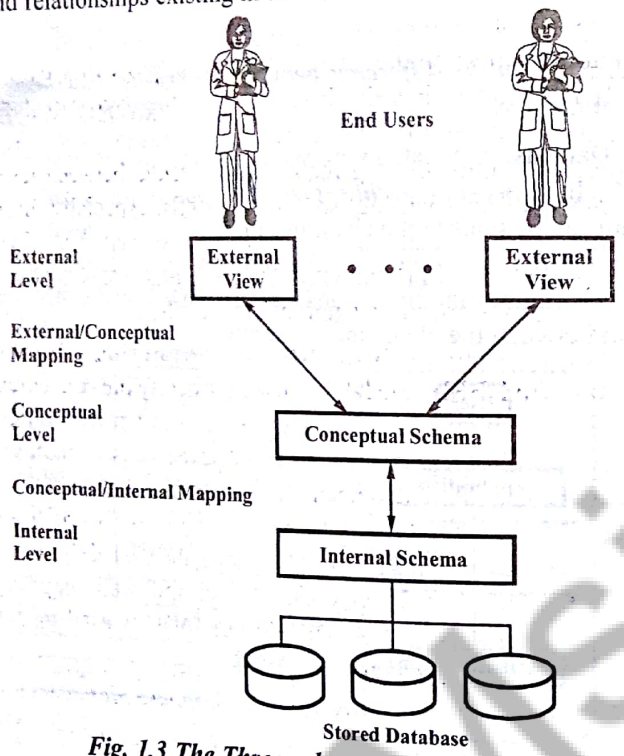


Fig. 1.3 The Three-schema Architecture

(i) Internal Level – The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

(ii) Conceptual Level – The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations and constraints. A high-level data model or an implementation data model can be used at this level.

(iii) External or View Level – The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of database from that user group. A high-level data model or an implementation data model can be used at this level.

Mapping between Schema Levels – The three schemas are only descriptions of data. The only data that actually lies is on the physical level. In a DBMS on the basis of three schema architecture, every user group refers only to its own external schema. Thus, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. Mappings are the process of transforming requests and results between levels. These mappings may be time-wasting, so some DBMSs, especially those that are meant to support small databases, do not support external views. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

Schema Definition Languages Supporting Three-schema Architecture – In DBMS, the DDL is used to specify the conceptual schema only, where a clear separation is maintained between the conceptual and internal levels. The storage definition language (SDL) is an another language used to specify the internal schema. The mappings between two schemas may be specified in either one of these languages. For a true three-schema architecture, we would require a third language, the view definition language (VDL) which is used to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas.

Q.14. Draw and discuss three levels of database management system. ***(R.G.P.V., May 2018)***

Or

With a neat sketch discuss the three schema architecture of a DBMS. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.13.

Q.15. Explain three level architecture. What are its objectives ? ***(R.G.P.V., June 2012)***

Ans. **Three Level Architecture** – Refer to Q.13.

Objectives of Three Level Architecture – The main objective of this three-level architecture is to separate each user's view of the database from the way the database is physically represented. This separation is required due

to several reasons. Some of the reasons are given below –

(i) Each user should be able to access the same data but should have a different customized view of data. Each user should be able to change the way he or she views the data and this change should not affect other users.

(ii) The database administrator (DBA) should be able to change the database storage structures without affecting the user's views.

(iii) The DBA should be able to change the conceptual structure of the database without affecting all users.

Q.16. What are the different levels of data abstraction ? Explain each of them. ***(R.G.P.V., Dec. 2016)***

Ans. There are three types of levels of data abstraction which are following here –

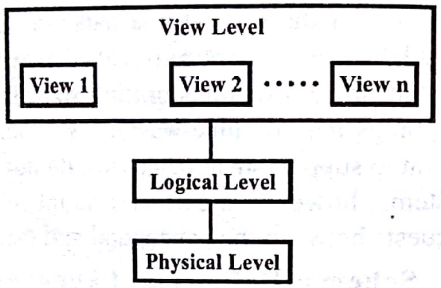


Fig. 1.4 Three Levels of Data Abstraction

(i) Physical Level – It is the lowest levels of abstraction which describe how the data are actually stored. Because physical level describe complex low-level data structures in detail.

(ii) Logical Level – It is the higher level of abstraction which describes what data are stored in the database, and what relationships exist among those data. Logical level describes a small number of relatively simple structures in terms of the entire database. Although the implementation of the simple structures at logical level may include complex physical level structures, the user of the logical level does not need to be aware of this complexity. Database administrators decide that, what information to keep in the database, and use the logical level of abstraction.

(iii) View Level – It is also higher level of abstraction which is describes only part of entire database. Because view level uses simpler structures, complexity and remain variety of information stored in a large database. The view level of abstraction exists to simplify their interaction with the system and system can provide many view for the same database.

DATA MODELS, SCHEMAS AND INSTANCES

Q.17. Define the term data models. ***(R.G.P.V., Dec. 2013)***

Or

What is a data model ? ***(R.G.P.V., Dec. 2015)***

Ans. A model is an abstraction process that hides superfluous details while highlighting details pertinent to the applications at hand. Data model is a mechanism that provides the abstraction for database applications. Data modeling is used to represent entities of interest and their relationships in the database. It allows the conceptualization of the association between various entities and their attributes. We can say that data model is a collection of conceptual tools to describe data, data relationships, data semantics and consistency constraints.

Q.18. Discuss the main categories of data models. ***(R.G.P.V., Nov./Dec. 2007, June 2010, 2011)***

Or

Explain various data models. ***(R.G.P.V., Dec. 2015)***

Or

What are the different data models present ? Explain in detail. ***(R.G.P.V., June 2015)***

Ans. One data model can be distinguished from other on the basis of the way relationship among data, that is defined and the way the data is conceptually defined. There are many data models, chosen as per need of the application. These are fallen in following categories –

(i) Object-based logical models (ii) Record-based logical models

(iii) Physical data models.

(i) Object-based Logical Models – These models are used in describing data at logical and view levels. They are characterized by the fact that they provide flexible structuring capabilities and allow data constraints to specify explicitly. This model emphasizes that, everything is object having a set of attributes. There are many data models in this category –

(a) Entity-relationship model (b) Object-oriented model

(c) Semantic data model (d) Functional data model.

(ii) Record-based Logical Models – These models are also used in describing data at logical and view levels. But in contrast to the object-based data models, they revolve around the records of the database and specify the overall structure of database, with the help of values of records. Record-based models are so named, since the database is structured in fixed-format records of several types.

Each record type defines a fixed number of fields or attributes and each field is usually of a fixed length. This simplifies the physical-level implementation of the database. The three most widely used record-based models are –

(a) Relational model (b) Network model (c) Hierarchical model.

(iii) Physical Data Models – This model is used to describe data at the lowest level i.e., to describe the behaviour of data at the disk level i.e., the

to several reasons. Some of the reasons are given below –

(i) Each user should be able to access the same data but should have a different customized view of data. Each user should be able to change the way he or she views the data and this change should not affect other users.

(ii) The database administrator (DBA) should be able to change the database storage structures without affecting the user's views.

(iii) The DBA should be able to change the conceptual structure of the database without affecting all users.

Q.16. What are the different levels of data abstraction ? Explain each of them. ***(R.G.P.V., Dec. 2016)***

Ans. There are three types of levels of data abstraction which are following here –

(i) Physical Level – It is the lowest levels of abstraction which describe how the data are actually stored. Because physical level describe complex low-level data structures in detail.

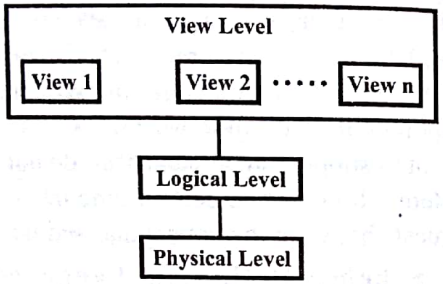


Fig. 1.4 Three Levels of Data Abstraction

(ii) Logical Level – It is the higher level of abstraction which describes what data are stored in the database, and what relationships exist among those data. Logical level describes a small number of relatively simple structures in terms of the entire database. Although the implementation of the simple structures at logical level may include complex physical level structures, the user of the logical level does not need to be aware of this complexity. Database administrators decide that, what information to keep in the database, and use the logical level of abstraction.

(iii) View Level – It is also higher level of abstraction which is describes only part of entire database. Because view level uses simpler structures, complexity and remain variety of information stored in a large database. The view level of abstraction exists to simplify their interaction with the system and system can provide many view for the same database.

DATA MODELS, SCHEMAS AND INSTANCES

Q.17. Define the term data models. ***(R.G.P.V., Dec. 2013)***

Or

What is a data model ? ***(R.G.P.V., Dec. 2015)***

Ans. A model is an abstraction process that hides superfluous details while highlighting details pertinent to the applications at hand. Data model is a mechanism that provides the abstraction for database applications. Data modeling is used to represent entities of interest and their relationships in the database. It allows the conceptualization of the association between various entities and their attributes. We can say that data model is a collection of conceptual tools to describe data, data relationships, data semantics and consistency constraints.

Q.18. Discuss the main categories of data models. ***(R.G.P.V., Nov./Dec. 2007, June 2010, 2011)***

Or

Explain various data models. ***(R.G.P.V., Dec. 2015)***

Or

What are the different data models present ? Explain in detail. ***(R.G.P.V., June 2015)***

Ans. One data model can be distinguished from other on the basis of the way relationship among data, that is defined and the way the data is conceptually defined. There are many data models, chosen as per need of the application. These are fallen in following categories –

(i) Object-based logical models (ii) Record-based logical models

(iii) Physical data models.

(i) Object-based Logical Models – These models are used in describing data at logical and view levels. They are characterized by the fact that they provide flexible structuring capabilities and allow data constraints to specify explicitly. This model emphasizes that, everything is object having a set of attributes. There are many data models in this category –

(a) Entity-relationship model (b) Object-oriented model

(c) Semantic data model (d) Functional data model.

(ii) Record-based Logical Models – These models are also used in describing data at logical and view levels. But in contrast to the object-based data models, they revolve around the records of the database and specify the overall structure of database, with the help of values of records. Record-based models are so named, since the database is structured in fixed-format records of several types.

Each record type defines a fixed number of fields or attributes and each field is usually of a fixed length. This simplifies the physical-level implementation of the database. The three most widely used record-based models are –

(a) Relational model (b) Network model (c) Hierarchical model.

(iii) Physical Data Models – This model is used to describe data at the lowest level i.e., to describe the behaviour of data at the disk level i.e., the

MSInventer

way the data and data relationships are maintained while storing them on the disk. This decided the way the DBMS is going to use secondary storage devices, for storing and accessing database. The widely used these data models are –

(a) Unifying model (b) Frame memory model.

Q.19. Discuss the role of a high level data model in the database design process. ***(R.G.P.V., June 2010)***

Ans. Conceptual or high-level data models offer concepts which are close to the way many users perceive data. Concepts like relationships, entities and attributes are used by conceptual data models. A relationship among two or more entities represents an action among the entities. For instance, a works-on relationship between an employee and a project. An entity represents a real-world concept or object like a project or an employee, which is described in the database. An attribute represents few properties of interest which further defines an entity, like the employee id and designation.

Q.20. Write short note on hierarchical data model. ***(R.G.P.V., Dec. 2011, 2013)***

Ans. A hierarchical database consists of a collection of records that are connected to each other through links. Each record is a collection of attributes (fields), each of which has only one data value. A link is an association between two records.

Suppose a database shows a *customer_account* relationship in banking system. Here, the customer and account, are two record types. The customer record consists of three fields, customer_name, customer_street and customer_city. Similarly, the account record consists of account_number and balance.

Fig. 1.5 shows a database sample, in which customer Hayes has account A-102, Johnson has account A-101 and A-201 and Turner has account A-305.

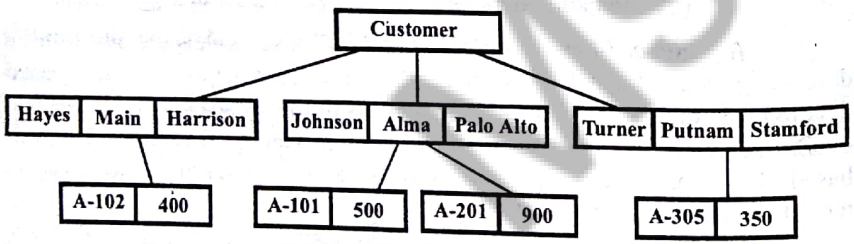


Fig. 1.5 Database Sample

Here, set of all customer and account records is organized in form of a rooted tree, where root of tree is a dummy node.

For a hierarchical database, a tree-structure diagram is a schema. Such diagram consists of two basic components, boxes that corresponds to record types and lines that corresponds to links. A tree-structure diagram serves the

same purpose of E-R diagram does i.e., it specifies overall logical structure of database. Suppose, we consider the E-R diagram of fig. 1.6.

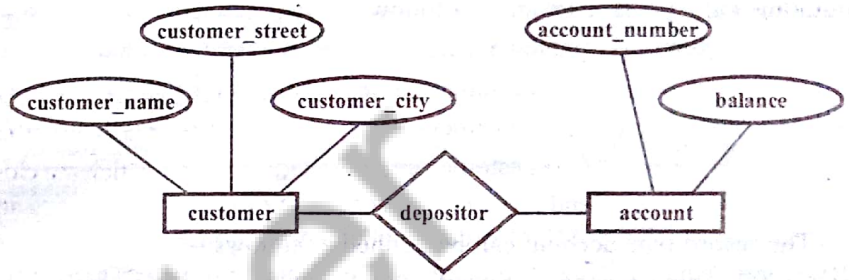


Fig. 1.6 E-R Diagram

In fig. 1.6, E-R diagram consists of two entity sets i.e., ***customer*** and ***account*** related through a binary, one-to-many relationship depositor with no descriptive attributes. This diagram specifies, a customer may have several accounts, but an account can belong to only one customer. Fig. 1.7 shows its corresponding tree-structure diagram.

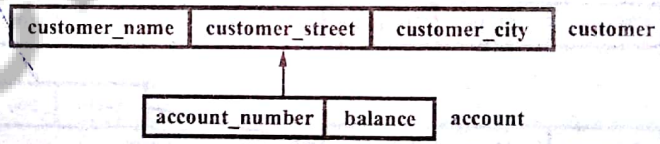


Fig. 1.7 Tree-structure Diagram

In this, the record type customer corresponds to the entity set ***customer***. It has three fields i.e., customer_name, customer_street and customer_city. The record type account corresponds to the entity set ***account***. It has two fields i.e., account_number and balance. And the relationship ***depositor*** is replaced with the link depositor, with an arrow pointing to customer record type.

Q.21. What are data models ? Discuss briefly any one data model. ***(R.G.P.V., Dec. 2016)***

Ans. **Data Models** – Refer to Q.17.

Hierarchical Data Model – Refer to Q.20.

Q.22. Discuss the basic concepts of network data model. ***(R.G.P.V., June 2015)***

Or

Write short note on network data model. ***(R.G.P.V., Dec. 2011, 2013)***

Ans. The network data model represents data for an entity set by a logical record type. A network database consists of a collection of records connected to one another through links. In many respects, a record is similar to an E-R model. Each record is a collection of fields (attributes), each of which has only one data value. A link is an association between two records.

Lets consider a database that represents a customer-account relationship in a banking system. Two record types, customer and account are here. We can define record type customer as follows –

```
type customer = record
                    customer-name : string;
                    customer-street : string;
                    customer-city : string;
                end
```

The record type account can be defined as follows –

```
type account = record
                    account-number : string;
                    balance : integer;
                end
```

Fig. 1.8 shows a database sample, in which Hayes has account A-102, Johnson has accounts A-101 and A-201, Turner has account A-305.

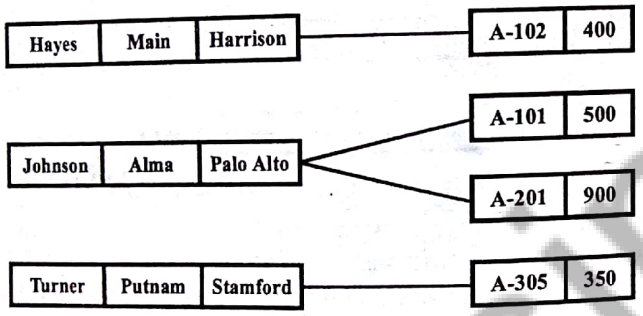


Fig. 1.8 Database Sample

To represent the design of a network database, a data-structure diagram is a schema. Such a diagram involves two components i.e., boxes correspond to record types and lines correspond to links. This diagram serves the same purpose as E-R diagram i.e., specifies overall logical structure of the database. Let's consider on fig. 1.9.

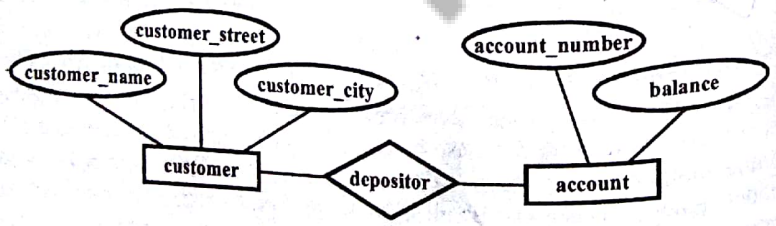


Fig. 1.9 E-R Diagram

Here, E-R diagram consists of two entity sets i.e., ***customer*** and ***account***, related through many-to-many relationship *depositor* with no descriptive attributes. Diagram specifies, a customer can have several accounts but an account belongs only one customer. Its corresponding data-structure diagram is shown in fig. 1.10.

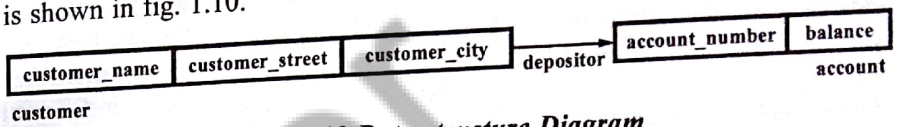


Fig. 1.10 Data-structure Diagram

Here, record type customer corresponds to entity set ***customer*** and it has three fields i.e., customer_name, customer_street and customer_city. The record type account corresponds to entity set ***account*** and it has two fields, account_number and balance. The relationship ***depositor*** is replaced with link despositor.

If relationship ***depositor*** were one-to-many from ***customer*** to ***account***, then link depositor would have an arrow pointing to record type customer. Similarly, if relationship ***depositor*** were one-to-one, then link depositor would have two arrows, one pointing to record type account and other one pointing to record type customer.

Q.23. Discuss the concepts of network and hierarchical model. ***(R.G.P.V., June 2013)***

Ans. **Network Model** – Refer to Q.22.

Hierarchical Model – Refer to Q.20.

Q.24. Write short note on web based data model.(R.G.P.V., Dec. 2017)

Ans. Most websites rely on some kind of database to organize and present data to users. Whenever someone uses the search functions on these sites, their search terms are converted into queries for a database server to process. Typically, middleware connects the web server with the database.

The broad presence of databases allows them to be used in almost any field, from online shopping to micro-targeting a voter segment as part of a political campaign. Various industries have developed their own norms for database design, from air transport to vehicle manufacturing.

The main part of information and data determined on the web are mainly being recovered and stored with the use of RDBMS. Relational database is made up of relational data model. A data model is specified by its structure, constraints and functional support. Constraints such as primary and foreign keys for specifying data uniqueness and relationship among supporting contents of a relation. Transforming relational data model into semantic web based model needs careful mapping to both models. Many techniques and tools have been presented to help by giving the ways to explore relational data

transformation to become compatible to semantic web based systems. While, transformation task attempts to map two distinct data models to come to an agreement. Some of the state of art methods introduced for transformation involve direct mapping, R2O, eD2R, relational.OWL, D2RQ, Triplify, R2RML, and R3M. Yet there exist difficulties in clearly gaining results with high performance and compatibility. Various researchers have attempted various methods to get the needed outcome of reaching to a next generation web which is the semantic web. Diverse data analysts have worked with different ways, to reduce the problems, and to satisfy the needed results, via the use of cutting edge web technology which is semantic web. Usually, data is stored in the form of relational database in web applications. Compatibility issues arise if same data is transformed from RDB to semantic web based system. It needs looking into the weaknesses produced during the transformation process from RDB into the semantic web. For evolving data keeping changes intact is hard and difficult to sustain. To get this, data mapping may be employed to understand their differences at the level of data types. Then mapping is done using XML based data structure as their intermediate data presenter. To map up common features found in both data models of RDB and semantic web based schema using either form of XML like document type definition (DTD) or XML schema like an intermediate that will help in improving transformation results. For data transformation, these data mappings can further help in gaining better compatibility options.

Q.25. Discuss the concept of relational data model in brief.

Ans. Relational data model is based on the collection of tables. The user of the database system may query these tables, insert new tuples, delete tuples and update tuples. To express these operations, there are several languages.

The relational data model, like all data models, consists of three basic components –

(i) A set of domains and a set of relations
(ii) Operations on relations
(iii) Integrity rules.

In relational database systems, attributes correspond to fields. For given application an attribute may only be allowed to take a value from a set of permissible values. This set of allowable values for the attribute is the domain of the attribute. A domain may be structured or unstructured.

The unstructured (atomic) domains are general sets, such as sets of integers, real numbers, character strings and so on. These are sometimes referred to as application-independent domains. Structured (composite) domains can be specified as consisting of nonatomic values. The domain for attribute address, for instance, that specifies street number, street name, city, state and zip or postal code is considered as composite domain.

An entity type having n attributes can be represented by an ordered set of these attributes called an n-tuple. If these n attributes take values from domains $D_1, D_2, ..., D_n$. The representation of entity must then be a member of the set $D_1 \times D_2 \times ..... \times D_n$, as resulting set of this cartesian product has all possible ordered n-tuples.

Conceptually, a relation is represented as a table. Each column of table represents an attribute and each row represents a tuple of the relation. In relational model, we represent the entity by a relation and use a tuple to represent an instance of the entity. Different instances of an entity type are distinguishable.

Codd defined a "relationally complete" set of operations and the collection of these, that take one or more relations as their operand(s), forms the basis of relational algebra. To manipulate relations, there are number of operations. Relations can be derived from other relations or a number of relations can be combined to define a new relation. The transformation of relations is useful in getting results from database.

Relational model involves two integrity rules. These implicitly or explicitly define the set of consistent database states, or changes of states or both. Other integrity constraints can be specified in terms of dependencies during database design. First integrity rule is concerned with primary key values. If any attribute of a primary key were permitted to have null values then attributes in key must be nonredundant, the key cannot be used for unique identification of tuples. This rule is also referred to as entity rule. Second integrity rule is concerned with foreign keys i.e., with attributes of a relation having domains that are those of the primary key of another relation. This rule is also referred as referential integrity rule. Fig. 1.11 shows representation of a relation as a table.

APPLICANT –

Name	Age	Profession
John Doe	55	Analyst
Mirian Taylor	31	Programmer
Abe Malcolm	28	Receptionist
Adrian Cook	33	Programmer
Liz Smith	32	Manager

Fig. 1.11 Example Representation of a Relation as a Table

Q.26. Write characteristics of relational model. (R.G.P.V., June 2014)

Ans. Relational databases have some particular characteristics, which are as follows –

(i) Ordering of rows
(ii) Ordering of columns
(iii) Atomicity of values
(iv) Uniqueness of rows.

Q.27. Discuss the various features of relational data model in detail. (R.G.P.V., Dec. 2011)

Ans. Refer to Q.26 and Q.25.

Q.28. Explain data model and its types. ***(R.G.P.V., June 2007)***

Or

Define data model. Explain the different types of data models with relevant examples. ***(R.G.P.V., Dec. 2012)***

Ans. **Data Model** – Refer to Q.17.

Its Types – Refer to Q.18, Q.20, Q.22 and Q.25.

Q.29. Compare the procedure of Insert, Delete and Update operations in network and relational model. ***(R.G.P.V., Dec. 2010)***

Ans. Insert, delete and update operations in network model are done as follows –

(i) The STORE Command – This command is used to insert a new record. Before issuing a STORE, we must first set up the UWA variable of the corresponding record type so that its field values contain the field values of the new record. The UWA (user work area) is the set of local variables in the program needed by the host programming language to holds the record of different record types so that their contents can be manipulated by the host program. For instance, to insert a new EMPLOYEE record for David F. Smith, we can prepare the data in the UWA variables, then issue

$ STORE EMPLOYEE;

The result of the STORE command is insertion of the current contents of the UWA record of the specified record type into the database.

(ii) The ERASE and ERASE ALL Commands – To delete a record from the database, we first make that record the CRU and then issue the ERASE command. For example, to delete the EMPLOYEE record –

EMPLOYEE.SSN:='261245897';

$ FIND ANY EMPLOYEE **USING** SSN;

if DB_STATUS = 0 then **$ ERASE** EMPLOYEE;

A variation of the ERASE command, ERASE ALL which allows the programmer to remove a record and all records owned by it directly or indirectly. This means that all member records owned by the record are deleted. In addition, member records owned by any of the deleted records are also deleted, down to any number of repetitions.

(iii) The MODIFY Command – The MODIFY command changes some of the field values of a record. Insert, delete and update in relational model are done as follows –

(a) Insertion – For inserting data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. Obviously, the attribute values for inserted tuple must be members

of the attribute's domain. Similarly, tuples inserted must be of the correct arity. The relational algebra denotes an insertion through –

$$r \leftarrow r \cup E$$

where r is a relation and E is a relational algebra expression. We express the insertion of a single tuple by letting E be a constant relation containing one tuple.

(b) Deletion – We can express a delete request in the similar way as a query. However, instead of displaying tuples to the user, we remove the selected tuples from the database. We can delete only whole tuples. We cannot delete values on only particular attributes. In relational algebra a deletion is denoted by

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.

(c) Updation – In some conditions, we may wish to change a value in a tuple without changing all values in a tuple. We can use the generalized-projection operator to do this task –

$$r \leftarrow \pi_{F_1, F_2, \ldots, F_n}(r)$$

where each F_i is either the ith attribute of r, if the ith attribute is not updated, or if the attribute is to be updated, F_i is an expression, involving only constants and the attributes of r, that gives the new value for the attribute.

If we want to choose some tuples from r and to update only them, we can use the following expression –

$$r \leftarrow \pi_{F_1, F_2, \ldots, F_n}(\sigma_P(r)) \cup (r - \sigma_P(r))$$

Here, P denotes the selection condition that chooses which tuples to update.

Q.30. Compare and contrast the features of hierarchical, network and relational data models. What business needs led to the development of each of them ? ***(R.G.P.V., Dec. 2009)***

Or

Compare among hierarchical, network and relational data model. ***(R.G.P.V., June 2011)***

Ans. Having designed an E-R diagram for a database application, the relational model is relatively straightforward. Each entity type in E-R diagram is represented by a relation where in each attribute of the entity becomes attribute of the relation. Each instance of entity is represented by a tuple of the relation. A weak entity can be represented by a relation but must include primary key of identifying strong entity. In E-R diagram, each relationship is represented by a relation, the attribute of this relation being the primary keys of entities involved in relationship plus the attributes of relationship. Each instance of relationship set among entities is represented by a tuple of this relation.

MSinventer

When E-R diagram is converted into network model, it will be accomplished as follows. Each entity type in E-R diagram is represented by a record type and each instance of entity is represented by record occurrence. A weak entity is represented as a separate record type. In this, identifying relationship is represented as a set type wherein record type corresponding to weak entity type forms the member and record type corresponding to strong entity is owner. A 1 : 1 or 1 : N relationship is represented as a set type. M : N relationship needs to introduce immediate record type. This record type is a common member in two set types, one of which is owned by each of record types involved in M : N relationship.

When we convert E-R diagram into a hierarchical model, it is accomplished as follows. Each entity type in E-R diagram is represented by a record type. A 1 : N relationship is represented as a hierarchy type where record type corresponding to one side of relationship is the parent. A weak entity is represented as a separate record type. This record type becomes a dependent in a hierarchy where record type corresponding to strong entity in identifying relationship, is the parent. M : N relationship needs to introduce duplicate record types or using multiple hierarchies and to introduce virtual records.

In network model, it is possible that many identical occurrences of same logical record type could exist. These multiple identical occurrences are distinguished by their membership in different occurrences of sets. Similarly, in hierarchical model, identical occurrences of a record type are distinguished by their associations with different ancestor record type occurrences. The tuples of a relation are, however, unique because if the relation represents a relationship between entities, the relationships between occurrences of entities are explicitly recorded in tuples by inclusion of corresponding primary keys.

The relational model allows for a fairly straightforward method of selecting certain entities or relationships. This is done by selecting those tuples of relation corresponding certain selection conditions.

In network model, the selection operation on a record type is similar to that in the relational model. However, the operation corresponding to the join operation of relational model is handled differently. This involves following owner-to-member or the member-to-owner pointers.

The selection operations for record type corresponding to the root of a hierarchical tree are similar to operations for its counterpart in the relational and network models. As in case of network model, we have to traverse pointers from parent to child since there is no method of travelling from descendant to parent. However, a virtual scheme using a virtual record concept introduces reverse-navigation facility.

The process of joining relations in case of relational model or following the pointers from owner to member, from member to owner, or from parent to child is known as navigating through the database. Navigation through

relations that represent M : N relationship is just as simple as through a 1 : M relationship. This leads us to conclude that it is easier to specify how to manipulate a relational database than a network or hierarchical one. This in turn leads to a query language for the relational model that is correct, clear and effective in specifying the required operations.

The join operation is inherently inefficient and demands a considerable amount of processing and retrieval of unnecessary data. The structure for network and hierarchical models can be implemented efficiently. Such implementation would mean that navigating through these databases, through awkward, requires the retrieval of relatively little unnecessary data.

Q.31. What are different types of data models ? Explain relational data model with the help of some suitable example and compare with network data model. ***(R.G.P.V., June 2016)***

Ans. Refer to Q.18, Q.20, Q.22, Q.25 and Q.30.

Q.32. What are the advantages and disadvantages of relational database model, hierarchical database model and network database model ?

Ans. **Relational Database Model –**

Advantages – Advantages of the relational database model are –

(i) Structural Independence – Relational model does not depend on navigational data access system thus freeing the database designers, programmers and end users from learning details of data storage. Changes in database structure do not affect data access. When it is possible to make change to database structure without affecting DBMS's capability to access data. So it is structural independence.

(ii) Conceptual Simplicity – It is simple at conceptual level. Since relational data model frees designer from physical data storage details designers can concentrate on logical view of database.

(iii) Design, Implementation, Maintenance and Usage Ease – This model achieves both data independence and structural independence making database design, maintenance, administration and usage much easier than other models.

Disadvantages – Disadvantages of the relational database model are –

(i) Hardware Overheads – This system hides the implementation complexities and the physical data storage details from users. To do this i.e., to make things easier for users, this system needs more powerful hardware i.e., computers and data storage devices.

(ii) Ease of Design can Lead to Bad Design – It is easy-to-design and use system. Users need not know complex details of physical data storage. They need not know how data is actually stored to access it. This ease design

and use can lead to the development and implementation of very poorly designed systems. As database grows, poorly designed database will slow the system and will result in performance degradation and data corruption.

(iii) Information Island Phenomenon – The relational database systems are easy to implement and use. This will create a situation where too many people or departments will create their on databases and applications. These information islands will prevent information integration which is essential for smooth and efficient functioning of organization. These individual databases will also create problems like inconsistency, data duplication, redundancy and so on.

Hierarchical Database Model –

Advantages – Advantages of the hierarchical database model are –

(i) Simplicity – Since database is based on hierarchical structure, the relationship between various layers is logically simple. Thus, design of hierarchical database is simple.

(ii) Data Security – This model was first database model that offered data security which is proved and enforced by DBMS.

(iii) Data Integrity – Since this model is based on parent/child relationship, there is always a link between parent segment and child segments under it. Child segments are always automatically referenced to its parents, this model promotes data integrity.

(iv) Efficiency – This model is very efficient one when the database has a large number of 1 : n relationships (one-to-many) and when users need large number of transactions, using data whose relationships are fixed.

Disadvantages – Disadvantages of the hierarchical database model are –

(i) Implementation Complexity – Although this model is conceptually simple and easy to design it is quite complex to implement. The designers should have very good knowledge of physical data storage characteristics.

(ii) Database Management Problems – If we make any change in database structure in hierarchical database, then we need to make necessary changes in application programs that access database. Thus, maintaining database and applications can become very difficult.

(iii) Programming Complexity – Due to structural dependence and navigational structure, application programmers and end users must know precisely how the data is distributed physically in database in order to access data. This needs knowledge of complex pointer systems that is often beyond the grasp of users.

(iv) Lack of Structural Independence – Structural independence exists when changes to the database structure does not effect DBMS's ability to access data. To navigate to the different systems, hierarchical, systems use

physical storage paths. So application programmers should have good knowledge of relevant access paths to access data. Thus, if physical structure is changed the applications will also have to be modified.

(v) Implementation Limitation – In this model, many common relationships do not conform to the 1: n format. The many-to-many (n : n) relationships which are more common in real life are very difficult to implement in this model.

Network Database Model –

Advantages – Main advantages of the network database model are –

(i) Conceptual Simplicity – This is conceptually simple and easy to design.

(ii) Capability to Handle more Relationship Types – This model can handle one-to-many (1 : n) and many-to-many (n : n) relationships, which is a real help in modelling the real life situations.

(iii) Ease of Data Access – Data access is easier and flexible than in hierarchical model. An application can access owner record and all member records with in a set and if member in the set has two owners then one can move from one owner to another.

(iv) Data Integrity – This model does not allow a member to exist without an owner. Thus a user must first define the owner record and then member record. This ensures data integrity.

(v) Data Independence – This model is better than hierarchical model in isolating the programs from complex physical storage details. This to a certain extent, ensures that changes in data characteristics do not need changes to application programs.

Disadvantages – There are following disadvantages –

(i) System Complexity – This model provides navigational access to data in which data are accessed one record at a time. This navigational data access mechanism makes system implementation very complex and consequently data administrators, database designers, programmers and even end users should be familiar with internal data structures in order to access data. In other words, this model cannot be used to create a user friendly database management system.

(ii) Absence of Structural Independence – Since data access method in this model is a navigational system, making structural changes to the database is very difficult in most cases and impossible in some cases. If changes are made to the database structure then all application programs need to be modified before they can access data. Thus even though this model succeeds in achieving data independence, it still fails to achieve structural independence.

MSU Meerut

Q.33. Write any two advantages of network data model. ***(R.G.P.V., June 2016)***

Ans. Refer to Q.32.

Q.34. Explain the advantages of relational model and also explain the data control facilities in relational model. ***(R.G.P.V., Dec. 2011, 2013)***

Ans. **Advantages of Relational Model** – Refer to Q.32.

The data control facility gives features for one user to authorize other users to access his data. This facility also permits assertions to be made about data integrity.

Q.35. Differentiate between hierarchical, network and relational data models by taking examples. Also discuss their advantages and disadvantages. ***(R.G.P.V., June 2013)***

Or

Compare hierarchical, network and relational data model. Also, discuss their various advantages and disadvantages. ***(R.G.P.V., June 2014)***

Ans. **Differences between Data Models** – Refer to Q.30.

Advantages and Disadvantages of Data Models – Refer to Q.32.

Q.36. Describe the basic features of the relational data model. Discuss their advantages and disadvantages and importance to the end user and the designer. ***(R.G.P.V., Dec. 2009)***

Ans. **Basic Features of Relational Data Model** – Refer to Q.26 and Q.25.

Advantages and Disadvantages of Relational Data Model – Refer to Q.32.

Importance to the End-user and Designer – A relational table stores a collection of related entities. In this respect, the relational database table resembles a file. But there is one crucial difference between a table and a file – a table yields complete data and structural independence because it is a purely logical structure. How the data are physically stored in the database is of no concern to the user or the designer.

Another reason for the relational data model's rise to dominance is its powerful and flexible query language. For most relational database software, the query language is structured query language (SQL). SQL is a 4GL that allows the user to specify what must be done without specifying how it must be done. The RDBMS uses SQL to translate user queries into instructions for retrieving the requested data. SQL makes it possible to retrieve data with far less effort than any other database or file environment.

From an end-user perspective, any SQL based relational database application involves three parts –

(i) The end-user Interface – Basically, the interface allows the end-user to interact with the data.

(ii) A Collection of Tables Stored in the Database – In a relational database, all data are perceived to be stored in tables. The tables present the data to the end user in a way that is easy to understand.

(iii) SQL Engine – Largely hidden from the end user, the SQL engine executes all queires, or data requests.

Q.37. Explain the term database schema. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. Schema is the logical description of entire database. The overall design of the database is called database schema and it is changed rarely. Database systems have several schemas, partitioned according to the levels of abstraction. At lowest level is physical schema, at intermediate level is logical schema and at highest level is subschema or external schema. Generally, database systems support one physical schema, one logical schema and several subschemas. Basically, internal schema describes how the data are actually stored on disk. Conceptual schema describes the structure of the database to database designer. External schema or subschema describes the structure of database to end users. An end user gets the view of the database via external schema.

Q.38. What is meant by instance of the database schema ? ***(R.G.P.V., June 2014)***

Ans. Generally, the database is changed over time, as data is inserted, deleted or updated. The state of database at any particular moment of time i.e. the collection of data and information along with various objects definition is called the ***instance*** of the database. In database terminology, the instance of schema is a database.

Q.39. Define instance and schema. ***(R.G.P.V., Dec. 2015)***

Or

Write short notes on schemas and instances. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.38 and Q.37.

MSIU Meerut

DATA INDEPENDENCE, FUNCTIONS OF DBA AND DESIGNER

Q.40. Write short note on data independence. ***(R.G.P.V., June 2007)***

Or

Explain the term data independence. ***(R.G.P.V., Nov./Dec. 2007)***

Or

What is data independence and why it is important ?

(R.G.P.V., June 2016)

Ans. The term "data independence" can be explained easily with the three-schema architecture as follows – Data independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. The two types of data independence are –

STUDENT	Name	Student Number	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	Course Name	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structure	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

SECTION	SectionIdentifier	CourseNumber	Semester	Year	Instructor
	85	MATH2410	Fall	98	King
	92	CS1310	Fall	98	Anderson
	102	CS3320	Spring	99	Knuth
	112	MATH2410	Fall	99	Chang
	119	CS1310	Fall	99	Anderson
	135	CS3380	Fall	99	Stone

GRADE_REPORT	Student Number	SectionIdentifier	Grade
	17	112	B
	17	119	C
	8	85	A
	8	92	A
	8	102	B
	8	135	A

PREREQUISITE	CourseNumber	PrerequisiteNumber
	CS3380	CS3320
	CS3380	MATH2410
	CS3320	CS1310

Fig. 1.12 An Example of a Database that Stores Student Records and their Grades

(i) Logical Data Independence – It is the capacity to change the conceptual schema without having to change external schemas or application programs. The conceptual schema can be changed to expand the database, or to reduce the database. In the latter case, external schemas that refer only to the remaining data should not be affected. For example, the external schema of fig. 1.13 (a) should not be affected by changing the GRADE_REPORT file shown in fig. 1.12. into the one shown in fig. 1.14 (a). Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence.

(a)

TRANSCRIPT	StudentName	Student Transcript				
		CourseNumber	Grade	Semester	Year	SectionId
	Smith	CS1310	C	Fall	99	119
		MATH2410	B	Fall	99	112
	Brown	MATH2410	A	Fall	98	85
		CS1310	A	Fall	98	92
		CS3320	B	Spring	99	102
		CS3380	A	Fall	99	135

(b)

PREREQUISITES	CourseName	CourseNumber	Prerequisites
	Database	CS3380	CS3320
			MATH2410
	Data Structure	CS3320	CS1310

Fig. 1.13 Two Views Derived from Fig. 1.12

(a)

GRADE_REPORT	StudentNumber	StudentName	SectionIdentifier	CourseNumber	Grade
	17	Smith	112	MATH2410	B
	17	Smith	119	CS1310	C
	8	Brown	85	MATH2410	A
	8	Brown	92	CS1310	A
	8	Brown	102	CS3320	B
	8	Brown	135	CS3380	A

(b)

GRADE_REPORT	StudentNumber	StudentName	SectionIdentifier	CourseNumber	Grade
	17	Brown	112	MATH2410	B

Fig. 1.14

Not Inverted

(ii) Physical Data Independence – It is the capacity to change the internal schema without having to change the conceptual (or external) schemas. Changes to the internal schema may be needed because some physical files had to be reorganized – for example, by creating additional access structures – to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema. For example, providing an access path to improve retrieval of SECTION records in fig. 1.12 by Semester and Year should not require a query such as "list all sections offered in fall 1998" to be changed.

Importance of Data Independence – Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog. Data independence is accomplished because, when the schema is changed at some level, the schema at the next higher level remains unchanged, only the mapping between the two levels is changed. Hence, application programs referring to the higher-level schema need not to be changed.

Q.41. Discuss the three level architecture of DBMS. Explain how does it lead to data independence. ***(R.G.P.V., Dec. 2012)***

Ans. Refer to Q.13, and Q.40 under the heading 'importance of data independence'.

Q.42. Write differences between the logical and physical data independence. ***(R.G.P.V., Dec. 2006)***

Or

Differentiate between physical and logical data independence. ***(R.G.P.V., June 2013)***

Or

Define the two level of data independence. ***(R.G.P.V., Dec. 2014, June 2015)***

Ans. Logical data independence is the capacity to change the conceptual schema without having to change external schema or application programs.

In turn physical data independence is the capacity to change the internal schema without having to change the conceptual (or external) schemas.

Q.43. What is the difference between logical data independence and physical data independence ? Which one is harder to achieve ? Why ? ***(R.G.P.V., June 2011)***

Ans. **Difference** – Refer to Q.42.

By providing the external level or user view of the database, logical data independence is achieved. The application programs or users see the database as described by their respective external views. The DBMS offers a mapping from this view to the conceptual view. The sum of the community view of the database is the view at the conceptual level of the database. There is only a single conceptual view of a database but many external views. The users are only interested in those portion of the database which is described by their external view. It is an abstraction of the physically stored data and the user manipulates this abstraction.

Through the presence of the internal level of the database and the mappings or transformation from the conceptual level of the database to the internal level, physical data independence is achieved. Therefore, conceptual level to internal level mapping offers a means to go from the conceptual view to the internal view and from that place to the stored data in the database. A change is needed in the transformation functions between the physical and conceptual levels, if there is a need to change the type of physical device used or the file organization as a result of growth in the database or new technology. This change is required to maintain the conceptual level invariant. Alternating the physical database organization, however, can affect the response and efficiency of existing application programs. This may mean that while some application programs run faster, others can be slowed down. Regardless, no changes are required in the application programs themselves and they will run appropriately with the new physical data organization.

The physical data independence criterion requires that the conceptual level does not specify storage structures or the access methods used to retrieve the data from the physical storage medium. Making the conceptual schema physically data independent means that the external schema, which is defined on the conceptual schema, is in turn physically data independent.

Q.44. Explain the role of database administrator. (R.G.P.V., June 2007)

Or

Write short note on role of DBA. ***(R.G.P.V., Nov./Dec. 2007)***

Or

What is the role of DBA in Database management system ? ***(R.G.P.V., Dec. 2013)***

Ans. The ***data administrator*** (DA) is a person who makes the strategic and policy decisions regarding the data of the enterprise and the ***database administrator (DBA)*** is the person, who provides the necessary technical

MsInvert

support for implementing those decisions. Thus, DBA is responsible for the overall control of the system at a technical level.

Also, one of the main reasons for using DBMS is, to have central control of both the data and the programs that access those data. The person who has such central control over the system is called the database administrator (DBA).

Q.45. What are the main functions of database administration ? ***(R.G.P.V., June 2013)***

Or

Describe five main functions of a database administrator. ***(R.G.P.V., June 2008)***

Or

What are the responsibilities of database administrator ? Explain them in brief. ***(R.G.P.V., Dec. 2008)***

Or

Write down any two major responsibilities of a database administer. ***(R.G.P.V., Dec. 2014)***

Or

What are the functions of a DBA ? ***(R.G.P.V., June 2015)***

Or

Write any three functions of DBA. ***(R.G.P.V., June 2016)***

Ans. Some of the functions of database administrator are as follows –

(i) Schema Definition – The DBA creates the original database schema by executing a set of data definition statements in DDL.

(a) Defining the Conceptual Schema – It is the data administrator's job to decide exactly what information is to be held in the database. In other words, to identify the entities of interest to the enterprise and to identify the information to be recorded about those entities. This process is usually referred to as logical (sometimes conceptual database design). Once the data administrator has decided the contents of the database at an abstract level, the DBA will then create the corresponding conceptual schema, using the conceptual DDL. The object form of that schema will be used by DBMS in responding to access requests.

(b) Defining the Internal Schema – The DBA must also decide, how the data is to be represented in the stored database. This process is usually referred to as physical database design. Having done the physical design, the DBA must then create the corresponding storage structure definition, using the internal DDL.

(ii) Storage Structure and Access Method Definition – The DBA creates the appropriate storage structures and access methods by writing a set of definitions, which is translated by the data-storage and data-definition language compiler.

(iii) Defining Integrity Constraints – Data values stored in the database, must satisfy certain consistency constraints. For example, perhaps the number of hours an employee may work in 1 week, may not exceed a specified limit (say, 80 hours). Such a constraint must be specified explicitly by the database administrator.

(iv) Security – In large organization, many users access the database. But not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about various bank employees. They do not need access to information about customer accounts. So, DBA provides every user a unique ***password*** for system security.

(v) Granting of Authorization for Data Access – As explained above, granting of different types of authorization allows the database administrator to regulate which parts of the database various users can access. The authorization information is kept in a special system structure that is consulted by the database system, whenever access to the data is attempted in the system.

(vi) Internal Marketing – It is the business of DBA to ensure every user, that the database contains every useful data they want, in the desired format (conceptual schema). Effective internal marketing may reduce resistance to change and data ownership problems.

(vii) Managing Data Dictionary – Data dictionary contains metadata, that describes the data and data processing units. It is useful tool for database administration and is used throughout the database system life cycle.

(viii) Monitoring DBMS Performance – It is DBA, who is responsible for designing DBMS in such a manner that is best for the organization. He/she may need to have regular maintenance activities that ensures the appropriate performance at all the time.

(ix) Selection of Hardware and Software – DBA is the only one who decides what are the hardware and software requirements of the system.

Selection is done, keeping in mind the financial condition of the firm and thus in the manner that ensures maximized performance with lowest cost. Selection criteria includes that, whether the selected hardware and software enables the firm to fight in the market with competitors or not.

(x) Data Backup and Recovery – DBA must ensure that DBMS is facilitated with various backup procedures and recovery schemas. This is must to ensure it due to any error, database fails, the data could be easily recovered and the system will be reformed in consistent state, with no or minimal loss of data.

Q.46. What is the difference between database user and database administrator ? Explain various functions of database administrator. ***(R.G.P.V., May 2018)***

Ans. **Database User** – End users are the people whose jobs need access to the database for querying, updating, and producing reports. The database primarily exists for their use. End users have several categories –

(i) Stand-alone Users – These users maintain personal databases. To maintain personal databases, these users use ready-made program packages. These program packages give easy to use menu or graphics-based interfaces.

Example – User of tax package which stores a variety of personal financial data for tax purposes.

(ii) Casual End Users – These users occasionally access the database but they may require several information each time. Casual end users use a sophisticated database query language to specify their requests and are usually middle or high level managers or other occasional browsers.

(iii) Naive or Parametric End Users – These users make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates – known as canned transactions which have been carefully programmed and tested. This type of users performs different tasks as follows –

(a) Bank tellers check account balances and post withdrawals and deposits.

(b) Reservation clerks for airlines, hotels and car rental companies check availability for a provided request and make reservations.

(c) At receiving stations, clerks for courier mail enter package identifications through bar codes and descriptive information via buttons to update a central database of received and in transit packages.

(iv) Sophisticated End Users – These users include engineers, scientists, business analysts and others who thoroughly familiarize themselves with the facilities of the DBMS as to implement their applications to meet their difficult needs.

Database Administrator – Refer to Q.44.

Functions of Database Administrator – Refer to Q.45.

Q.47. What are the responsibilities of a DBA ? If we assume that the DBA is never interested in running his or her own queries, does the DBA still need to understand query optimization ? Why ? ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.45.

A DBA needs to understand query optimization even if he is not interested in running his or her own queries because some of these responsibilities are related to query optimization. Unless the DBA understands the performance needs to widely used queries, and how the DBMS will optimize and execute these queries, good design and tuning decisions cannot be made.

Q.48. Why is the administration of a database system required ? What are the various functions associated with database administration ? ***(R.G.P.V., Dec. 2012)***

Ans. Refer to Q.44 and Q.45.

Q.49. Explain the responsibilities of the DBA and the database designers ? ***(R.G.P.V., June 2017)***

Ans. **Responsibilities of DBA** – Refer to Q.45.

Responsibilities of Database Designer – Database designer is responsible for defining the detailed database design, including tables, indexes, views, constraints, triggers, stored procedures and other database specific constructs needed to store, retrieve and delete persistent objects.

ENTITIES AND ATTRIBUTES, ENTITY TYPES, KEY ATTRIBUTES, RELATIONSHIPS, DEFINING, THE E-R DIAGRAM OF DATABASE

Q.50. Explain entity and attributes. ***(R.G.P.V., Dec. 2012)***

Or

Define entity and attributes. ***(R.G.P.V., Dec. 2013)***

Ans. **Entity** – An entity is a thing in the real world with an independent existence. An entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job or a university course.

Attributes – Each entity has attributes i.e., the particular properties that describe it. For example, an employee entity may be described by the employee's

name, age, address, salary and job. Each attribute of an entity set has a particular value. The set of possible values that a given attribute can have is called its ***domain***.

Q.51. What do you mean by entity types and entity sets ? Explain.

Ans. An ***entity type*** defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes. Fig. 1.15 shows two entity types, named EMPLOYEE and COMPANY, and a list of attributes for each. The collection of all entities of a particular entity type in the database at any point in time is called an ***entity set***. The entity set is referred to using the same name as the entity type. For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.

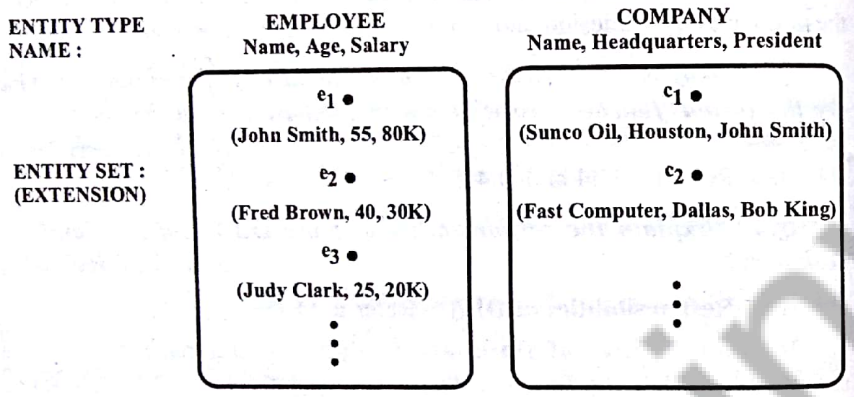


Fig. 1.15

Q.52. What is an entity type ? What is an entity set ? Explain the differences among an entity, an entity type and an entity set. ***(R.G.P.V., June 2010)***

Ans. Refer to Q.51 and Q.50.

Q.53. Discuss various types of attributes.

Or

Explain following –

(i) Simple and composite attributes

(ii) Single value and multivalue attributes

(iii) Derived attributes.

(R.G.P.V., Dec. 2012)

Ans. The types of attributes are as follows –

(i) Composite Versus Simple (Atomic) Attributes – Composite attributes are the attributes, which can be divided into subparts to represent more basic attributes with independent meanings. For example, an attribute ***name*** can be structured as a composite attribute consisting of first_name, middle_name and last_name. On the other hand, attributes that are not divisible are called simple or atomic attributes.

Composite attributes can form a hierarchy. For example, in composite attribute address, its component attribute street can be divided into three subparts – street_number, street_name, and apartment_number as shown in fig. 1.16.

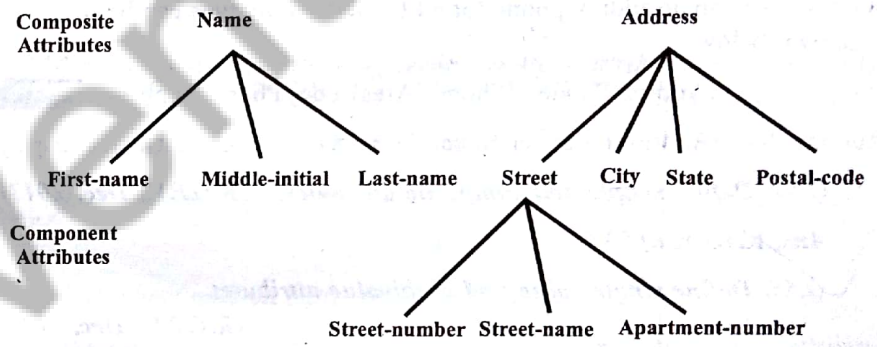


Fig. 1.16 Composite Attribute Customer_name and Customer_address

Composite attributes are useful to model situations, in which a user wish to refer to an entire attribute on some occasions and to only a component of the attribute on other occasions.

(ii) Single-valued Versus Multivalued Attributes – The attributes, that have a single value for a particular entity, are called single valued. For example, age is a single-valued attribute for a person entity.

On the other hand, an attribute can have a set of values for the same entity, are known as multivalued. For example, phone number attribute for an entity set employee may have zero, one or several phone numbers.

A multivalued attribute may have lower and upper bounds on the number of values allowed for each individual entity. For example, a bank may limit the number of phone numbers recorded for a single customer to two.

(iii) Stored Versus Derived Attributes – In some cases two (or more) attribute values are related. For example, the customer entity set

MsInventer

has an attribute age, which indicates the customer's age. If the customer entity set also has an attribute date_of_birth, we can calculate age from date_of_birth and the current date. Thus, the age is derived attribute. In this case, date_of_birth may be referred to as a ***base attribute*** or a ***stored attribute***. The value of derived attribute is not stored, but is computed within required.

(iv) Complex Attributes – Composite and multivalued attributes can be nested in an arbitrary way. We can represent nesting by grouping components of a composite attribute between parenthesis ( ) and separating the components with commas and by displaying multivalued attributes between braces { }. Such attributes are called ***complex attributes***. For example, if a person can have more than one residence and each residence can have multiple phones, an attribute addressphone for a PERSON entity type can be specified as shown below

{AddressPhone ({Phone (AreaCode, PhoneNumber)},
Address (StreetAddress (Number, Street, ApartmentNumber), City, State, Zip))}

Q.54. Define simple and composite attributes. ***(R.G.P.V., Dec. 2013)***

Ans. Refer to Q.53 (i).

Q.55. Define single value and multivalue attributes.

(R.G.P.V., Dec. 2013)

Or

Differentiate between single valued and multivalued.

(R.G.P.V., June 2016)

Ans. Refer to Q.53 (ii).

Q.56. With an example explain what a derived attribute is ?

(R.G.P.V., Dec. 2014)

Ans. Refer to Q.53 (iii).

Q.57. Define the following terms –

(i) Entity

(ii) Attribute

(iii) Multivalued attribute.

(R.G.P.V., June 2017)

Ans. (i) Entity – Refer to Q.50.

(ii) Attribute – Refer to Q.50.

(iii) Multivalued Attribute – Refer to Q.53 (ii).

Q.58. What is key attribute of an entity type ?

Ans. An important constraint on the entities of an entity type is the key or uniqueness constraint on attributes. An entity type usually has an attribute whose values are distinct for each individual entity in the collection. Such an attribute is called a key attribute, and its value can be used to identify each entity uniquely.

Q.59. What is a relationship ?

Ans. A ***relationship*** is an association among several entities. It defines the mechanism by which two or more entity types or entities are related to each other.

For example, we can define a relationship, that associates customer Hayes with loan L-15. This relationship specifies that Hayes is a customer with loan number L-15.

Q.60. Explain the following terms briefly – attribute, domain, entity-relationship. ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.50 and Q.59.

Q.61. How to describe the E-R diagram of database ?

Or

Write short note on entity-relationship model.

Ans. Entity-relationship model moves around three things – entity, attribute and relationship. This model is based on perception that consists of collection of objects called entities and every two entities are distinguished from other through their own set of properties. The relationship exists between these entities.

E-R diagram graphically expresses the logical structure of database (schema) and it uses –

(i) Rectangles, to represent entity set

(ii) Ellipses, to represent attributes

(iii) Diamonds, to represent relationships among entity sets

(iv) Lines, to show the links between entities and relationships.

Let us take an example, each person and bank account can be considered to be entities. Entities are described in database by some properties called ***attributes***. For example, attributes *account_number* and *balance* describe one particular account in a bank. A ***relationship*** is an association among several entities. For example, a depositor relationship

MsinVenter

associates a customer with each account that he/she has. An E-R diagram representing relationship between *customer*, *account* and *depositor* is shown in fig. 1.17.

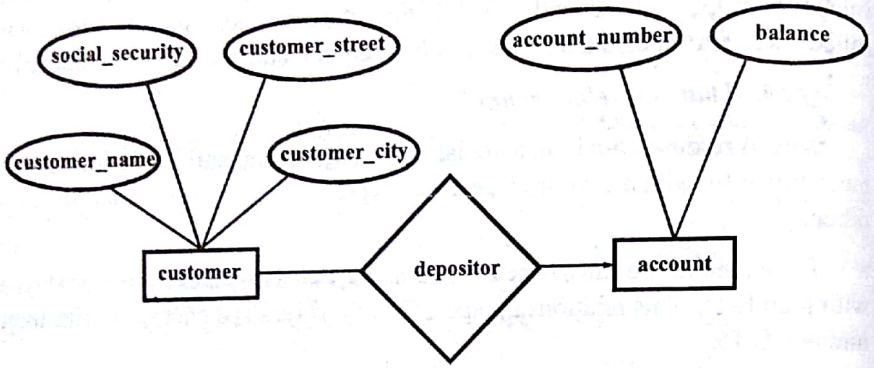


Fig. 1.17 E-R Diagram

⌘⌘

UNIT 2

RELATIONAL MODEL – STRUCTURE OF RELATIONAL DATABASES, DOMAINS, RELATIONS

Q.1. Explain relational databases and relational database schemas.

Ans. A relational database is defined as a collection of tables and each table is assigned a unique name as well as each table has contained similarly structure as in E-R databases. A table in a row represents a relationship among a set of values and a table is a collection of such relationship, relational databases or schemas is a close correspondence between the concept of table and mathematical concept of relation, because the relational data model also takes its name that is called as "relational database and relational database schemas".

Actually, a relational database contains many relations, with tuples in relations that are related in various ways. A relational database schema S is a set of relation schemas $S = \{R_1, R_2, ..., R_m\}$ and a set of integrity constraints. A relational database state DB of S is set of relation states $DB = \{r_1, r_2, ..., r_m\}$ such that each r_i is a state of R_i and such that the r_i relation states satisfy the specified integrity constraints.

For understanding database schema in a better way, we must differentiate between the database schema or the logical design of the database, and a database instance, which is a snapshot of the data in the database at a given instant in time.

The concept of relation schema corresponds to the programming language notion of type definition. It is convenient to give a name to a relation schema. We use the convention of using lowercase names for relations and names beginning with an uppercase letter for relation schemas.

Following this notion, we use account-schema to denote the relation schema for relation account.

Thus,

Account-schema = (branch_name, account_number, balance)

We denote the fact that account is a relation on Account-schema by account (Account-schema) as in table 2.1. In general, a relation schema comprises a list of attributes and their corresponding domains.

MsinVenkar

Table 2.1 The Account Relation

branch_name	*account_number*	*balance*
Downtown	A-101	500
Mianus	A-215	700
Perryridge	A-102	400
Round Hill	A-305	350
Brighton	A-201	900
Redwood	A-222	700
Brighton	A-217	750

Q.2. What are the design goals of relational databases ? Explain why each is desirable. ***(R.G.P.V., June 2013)***

Ans. The design goals are lossless-join decompositions, dependency preserving decompositions, and minimization of repetition of information. They are desirable so we can maintain an accurate database, check correctness of updates quickly, and use the smallest amount of space possible.

Q.3. Differentiate between intension and extension. ***(R.G.P.V., June 2008, Dec. 2008)***

Ans. We know that a set is well-defined collection of objects and it is represented by a list of elements (called members) or by the specification of some membership condition.

The ***intension*** of a set defines the permissible occurrences by specifying a membership condition.

The ***extension*** of the set specifies one of numerous possible occurrences by explicitly listing the set members.

These two methods of defining a set are illustrated as follows –

Intension of set G = {g | g is an odd positive integer less than 20}

Extension of set G = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}

Q.4. Explain relational schema, intension and extension. ***(R.G.P.V., June 2015)***

Ans. Refer to Q.1 and Q.3.

Q.5. What are the four broad categories of constraints ? ***(R.G.P.V., June 2015)***

Ans. Following are the four broad categories of constraints –

(i) Type Constraints – These constraints specify what is legal to a given type.

(ii) Attribute Constraints – These constraints specify that a specified column is of a specified type.

(iii) Instance Constraints – These constraints apply to a specific instance or condition.

(iv) Database Constraints – These constraints group two or more instance constraints.

Q.6. Write short note on integrity constraints. ***(R.G.P.V., June 2007)***

Or

Discuss the entity integrity and referential integrity constraints. Why is each considered important ? ***(R.G.P.V., Nov./Dec. 2007, June 2010, 2011, 2017)***

Or

Why are entity integrity and referential integrity important in database? Explain with suitable examples. ***(R.G.P.V., Dec. 2011, 2013)***

Or

What are the integrity rules ? Explain the significance of primary key and foreign key in employee database. ***(R.G.P.V., June 2012)***

Or

What are the different types of integrity constraints ? Explain by giving suitable example. ***(R.G.P.V., June 2015)***

Or

What are integrity constraints ? Explain various types of integrity constraints with suitable example. ***(R.G.P.V., May 2018)***

Ans. The term integrity refers to the accuracy or correctness of data in the database. Integrity constraints ensure that the changes made to the database by authorized users do not result in a loss of data consistency. Thus, the integrity constraints guard against accidental damage to the database. A database might be subject to any number of integrity constraints of arbitrary complexity.

Integrity constraints are specified on a database schema and are expected to hold on every database state of that schema. In addition to domain and key constraints, the relational model includes two general integrity rules – entity integrity and referential integrity. These integrity rules implicitly or explicitly define the set of consistent database states, or changes of state or both.

Integrity Rule 1 (Entity Integrity) – Integrity rule 1 is concerned with primary key values. The entity integrity constraint states that no primary key value can be null. This is because the primary key is used to identify individual tuples in a relation. Having null values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples have null for their primary keys, we cannot be able to distinguish them.

Consider the relation P(P) in fig. 2.1 (a). The attribute Id is the primary key for P (P). If null values (represented as @) are permitted, then the two tuples < @, smith > are indistinguishable, even though they may represent two different instances of the entity type employee. Similarly, the tuples < @, Lalonde > and < 104, Lalonde >, are also indistinguishable and may be referring to the same person.

MSinVenter

Integrity rule 1 specifies that instances of the entities are distinguishable and thus no prime attribute value may be null. This rule is also referred to as the entity rule. This rule can be formally stated as –

If attribute A of relation R(R) is a prime attribute of R(R), then A cannot accept null values.

P :

Id	Name
101	Jones
103	Smith
104	Lalonde
107	Evan
110	Drew
112	Smith

(a) Relation without Null Values

P :

Id	Name
101	Jones
@	Smith
104	Lalonde
107	Evan
110	Drew
@	Lalonde
@	Smith

(b) Relation with Null Values

Fig. 2.1

Integrity Rule 2 (Referential Integrity) – Integrity rule 2 is concerned with foreign keys i.e., with attributes of a relation having domains that are those of the primary key of another relation. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Consider the example of employee and their managers. Since each employee has a manager and as managers are also employees, we may represent managers by their employee numbers, if the employee number is a key of the relation employee. Thus, manager attribute represents the employee number of the manager. Note that it is referring to the primary key of the same relation. An employee can only have a manager who is also an employee, the chief executive officer (CEO) of the company can have himself or herself as the manager or may take null values.

Emp#	Name	Manager
101	Jones	@
103	Smith	110
104	Lalonde	107
107	Evan	110
110	Drew	112
112	Smith	112

Fig. 2.2 Foreign Keys

Q.7. What is referential integrity ? How it is implemented using SQL. ***(R.G.P.V., Dec. 2016)***

Ans. **Referential Integrity** – Refer to Q.6.

Implementation –

```
CREATE TABLE Works On {
eno CHAR (5) NoT NULL,
Pno CHAR (5) NoT NULL,
resp VARCHAR (20),
hours SMALLINT,
PRIMARY KEY (eno, Pno),
FOREIGN KEY (eno) REFERENCES
Emp ON DELETE NO ACTION ON
UPDATE CASCADE,
FOREIGN KEY (Pno) REFERENCES
Proj ON DELETE No ACTION ON
UPDATE CASCADE
};
```


Q.8. What is integrity constraint ? Explain the concept of referential integrity. ***(R.G.P.V., June 2014)***

Or

Explain referential integrity constraint with suitable example. ***(R.G.P.V., Dec. 2008)***

Or

Define the term referential integrity. ***(R.G.P.V., Dec. 2011)***

Ans. Refer to Q.6.

Q.9. What are the integrity rules ? Explain the concept of intension and extensions. ***(R.G.P.V., Dec. 2010)***

Ans. **Integrity Rules** – Refer to Q.6.

Intension and Extention – Refer to Q.3.

Q.10. Discuss the term domain in relational model.

Or

Define the term domain constraints. ***(R.G.P.V., Dec. 2013)***

Ans. A domain D is a set of atomic values. It means that each value in the domain is indivisible. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain to help in interpreting its values.

Some examples of domains are as follows –

(i) USA_phone_numbers – The set of 10-digit phone numbers valid in the united states.

(ii) Local_phone_numbers – The set of 7-digit phone numbers valid within a particular area code in the United States.

(iii) Social_security_numbers – The set of valid 9-digit social security numbers.

(iv) Names – The set of names of persons.

(v) Employee-ages – Possible ages of employees of a company between 15 and 80.

Q.11. Define domain integrity constraint, entity integrity constraint and referential integrity constraint. ***(R.G.P.V., Dec. 2017)***

Ans. **Domain Integrity Constraint** – Refer to Q.10.

Entity Integrity Constraint – Refer to Q.6.

Referential Integrity Constraint – Refer to Q.6.

Q.12. Explain with examples –

(i) Integrity constraints (ii) Relational schema

(iii) Domains (iv) Extension and intension of relation. ***(R.G.P.V., June 2013)***

Ans. (i) Integrity Constraints – Refer to Q.6.

(ii) Relational Schema – Refer to Q.1.

(iii) Domains – Refer to Q.10.

(iv) Extension and Intension of Relation – Refer to Q.3.

MSPWinter

Q.13. What do you mean by a relation ? Discuss it.

Ans. A relation consists of a homogeneous set of tuples. A relation (or relation state) r of the relation schema R (A_1, A_2, ..., A_n), also denoted by r (R), is a set of n-tuples r = {t_1, t_2, ..., t_m}. Each n-tuple t is an ordered list of n values t = < v_1, v_2, ..., v_n >, where each value v_i, $1 \le i \le n$, is an element of dom (A_i) or is a special ***null*** value.

The above definition of relation can be restated as follows –

A relation r (R) is a mathematical relation of degree n on the domains dom (A_1), dom (A_2), ..., dom (A_n), which is a subset of the cartesian product of the domains that define R –

$$r(R) \subseteq (dom(A_1) \times dom(A_2) \times ..... \times dom(A_n))$$

The cartesian product specifies all possible combinations of values from the underlying domain. Hence, if the number of values or cardinality of a domain D is denoted by | D |, and all domains are finite, then total number of tuples in the cartesian product is

$$| dom (A_1) | * | dom (A_2) | * .....* | dom (A_n) |$$

Q.14. What are the main properties of relation ?

Ans. The relation has some important properties which are as follows –

(i) There are no duplicate tuples
(ii) Tuples are unordered, top to bottom
(iii) Attributes are unordered, left to right
(iv) Each tuple contains exactly one value for each attribute.

Q.15. What are the characteristics of relation ? Explain domain, tuple, attributes, degree, cardinality of relation. ***(R.G.P.V., June 2013)***

Ans. **Characteristics of Relation** – Refer to Q.14.

Domain – Refer to Q.10.

Tuple – In relational model terminology, a row in a table, is known as a ***tuple***. An entity type having n attributes can be represented by an ordered set of these attributes called an ***n-tuple***. If these n attributes take values from the domains D_1, D_2, ... , D_n. The representation of the entity must then be a member of the set $D_1 \times D_2 \times .....\times D_n$, as the resulting set of this cartesian product contains all the possible ordered n-tuples.

A tuple is just like a record in conventional file systems and is used for handling entities and relationships between entities. Tuples are generally denoted by lowercase letters such as r, s, t, ... of the alphabet. An n-tuple t can be specified as

$$t = (a_1, ..., a_n)$$

where each a_i for $1 \le i \le n$ is a value in the D_i, and is the value of the attribute A_i in the tuple t. The order of the attributes is significant and fixed in the tuple representation. However, if we associate the attribute names with the corresponding values, we can relax the ordering requirement. Thus, the sets {(A_1 : a_1), (A_2 : a_2)} and {(A_2 : a_2), (A_1 : a_1)} are same. Therefore, a tuple can be viewed as a mapping from attribute names to values in the domains of the attributes.

Now, a tuple can be represented in a number of ways as follows –

$t = (a_1, ..., a_n)$ attribute value order must be constant

$t=(a_1,....,a_n|A_1,....,A_n)$ } attribute value can be
$t=(|A_1,....,A_n|\ |a_1,....,a_n|)$ } deduced from relative
$t=((A_1:a_1),....,(A_n:a_n))$ } ordering of the names
$t=((A_1|a_1),....,(A_n|a_n))$ } of the attribute

In the above formulations the a_i's are values drawn from D_i, the domain of A_i. The value of a tuple t over an attribute A_i is denoted as t [A_i], i.e., t[A_i] = a_i. It is known as the projection of the tuple t over A_i.

Attributes – An object or entity is characterized by its properties or attributes. In conventional file systems the term field refers to the smallest item of data with same practical meaning i.e., a field is used to capture some specific property of the object. In relational database systems, attributes correspond to fields. For each attribute there is a set of permitted values, called the domain of that attribute.

Degree – The degree also known as arity of a relation is number of attributes n of its relation schema.

Cardinality of Relation – The cardinality of a relation instance is the number of tuples in it.

Q.16. Explain following term –
(i) Domains (ii) Tuples (iii) Schemas.
(R.G.P.V., June 2016)

Ans. (i) Domains – Refer to Q.10.
(ii) Tuples – Refer to Q.15.
(iii)Schemas – Refer to Q.1.

RELATIONAL ALGEBRA – FUNDAMENTAL OPERATORS AND SYNTAX, RELATIONAL ALGEBRA QUERIES

Q.17. What do you understand by the term relational algebra ?

Ans. Relational algebra is a collection of operations to manipulate relations. The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. It specifies the operations to be performed on existing relations to derive result relations – Furthermore, it defines the complete scheme for each of the result relations.

Q.18. List out the various relational algebra operator. (R.G.P.V., Dec. 2014)

Ans. The relational algebra operations are usually divided into two groups. One group includes set operations from mathematical set theory; these are applicable because each relation is defined to be a set of tuples. Set operations include union, intersection, set difference and cartesian product. The other group consists of operations developed specifically for relational databases; these include select, project and join, among others.

The simplified definitions of these operators are as follows –

(i) Restrict/Select – It returns a relation containing all tuples from a specified relation that satisfy a specified condition.

(ii) Project – It returns a relation containing all tuples that remain in a specified relation after specified attributes have been removed.

(iii) Product – It returns a relation containing all possible tuples that are a combination of two tuples, one from each of two specified relations.

(iv) Union – It returns a relation containing all tuples that appear in either or both of two specified relations.

(v) Intersect – It returns a relation containing all tuples that appear in both of two specified relations.

(vi) Difference – It returns a relation containing all tuples that appear in the first and not the second of two specified relations.

(vii) Join – It returns a relation containing all possible tuples that are a combination of two tuples, one from each of two specified relations, such that the two tuples contributing to any given combination have a common value for the common attributes of the two relations.

(viii) Divide – It takes two unary relations one binary relation and returns a relation containing all tuples from one unary relation that appear in the binary relation matched with all tuples in the other unary relation.

Q.19. Explain the fundamental operations of relational algebra.

Or

Explain the basic relational algebra operation's with the symbol used and example for each. (R.G.P.V., Dec. 2013)

Ans. In the list of fundamental operations, select, project and rename are called the unary operations because they operate on one relation. The other three operations union, set difference and cartesian product operate on the pair of relations and are, therefore, called binary operations.

(i) The Select Operation – The ***select*** operation is used to select a subset of the tuples from a relation that satisfy a selection condition (or predicate). It means that select operation can be considered as a filter that keeps only those tuples that satisfy a qualifying condition. The symbol used for this operation is lowercase Greek letter sigma (σ) to denote selection.

Predicate appears as a sub-script to σ. For example, to select those tuples of the loan relation where the branch is "Perryridge", we can write as

$$\sigma_{\text{branch_name = "Perryridge"}} (\text{loan})$$

If the loan relation is as given in table 2.2.

Table 2.2 The Loan Relation

branch_name	*loan_number*	*amount*
Downtown	L-17	1000
Redwood	L-23	2000
Perryridge	L-15	1500
Downtown	L-14	1500
Mianus	L-93	500
Round Hill	L-11	900
Perryridge	L-16	1300

The result of above query can be given as shown in table 2.3.

Table 2.3 Result of $\sigma_{\text{branch_name = "Perryridge"}}$ (loan)

branch_name	*loan_number*	*amount*
Perryridge	L-15	1500
Perryridge	L-16	1300

In general, operators we can use for comparison in the selection predicate are $=, \neq, <, \leq, >, \geq$. We can also combine several predicates into a larger predicate using the connectives AND ($\wedge$) and OR ($\vee$), and NOT ($\neg$). For example, to find those tuples pertaining to loans of more than \$1200 made by the Perryridge branch, we write

$$\sigma_{\text{branch_name = "Perryridge"} \wedge \text{amount} > 1200} (\text{loan}).$$

So, by above discussion, we can say that select operation has following properties.

Notation : σ_p (R)
Purpose : Pick rows according to some criteria
Input : A table R
Output : Has the same columns as R, but only those rows of R that satisfies predicate P.

In selection predicate comparison between two attributes is also possible, for example consider the relation loan_officer that consists of three attributes customer_name, banker_name and loan_number, which specifies that a particular banker is the loan officer for a loan that belongs to some customer. To find all customers who have the same name as their loan officer, we can write

$$\sigma_{\text{customer_name = banker_name}} (\text{loan officer}).$$

But overall, this any comparison involving a null value evaluate to false because a null value indicates "value unknown or non-existent".

(ii) The Project Operation – If we think of a relation as a table, the project operation select certain columns from the table and discards the other columns. The general form of the project operation is

$$\Pi_{\text{<attribute list>}} (\text{R}).$$

MRIRVCE

where Π (pi) is the uppercase Greek letter to denote the project operation and <attribute list> is a list of attributes from the attributes of relation R. The result of the project operation has only the attributes specified in <attribute list> and in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>. For example, suppose we want to list all loan numbers and the amount of the loans, the query to list all numbers and the amount of the loan can be written as

$$\Pi_{loan_number,\ amount}\ (loan).$$

Table 2.4 Loan Number and the Amount of the Loan

loan_number	*amount*
L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-93	500
L-11	900
L-16	1300

Table 2.4 shows the relation that results from this query.

So by above explanation, project operation can be given as

Notation : $\Pi_L(R)$

Purpose : Pick the listed columns to output.

Input : A relation R

Output : A relation that has only those columns listed in L.

(iii) The Union Operation – The union operator is the usual set union of two relations R and S, which are union-compatible. Two relations are called union compatible, if they satisfy following two conditions –

(a) The relation R and relation S must be of same arity. That is, they must have the same number of attributes.

(b) The domains of the i^{th} attribute of R and the i^{th} attribute of S must be the same, for all i.

Here, R and S are temporary relations that are the result of relational-algebra expressions. For example, consider a query to find the names of all bank customers who have either an account or a loan or both. Since, *customer* relation does not have complete information i.e., information about account or loan, but it has only the information of customer i.e. *customer_name*, *customer_street* and *customer_city*, as shown in table 2.5.

Table 2.5 The Customer Relation

customer_name	*customer_street*	*customer_city*
Jones	Main	Harrison
Shyam	North	Rye
Hayes	Main	Harrison
Curry	North	Rye
Lindsay	Park	Pittsfield
Turner	Putnam	Stamford
Williams	Nassau	Princeton
Adams	Spring	Pittsfield
Jay	Alma	Palo Alto
Glenn	Sand Hill	Woodside
Brooks	Senator	Brooklyn
Green	Walnut	Stamford

So, we also need relation *borrower* and *depositor*, that have information about *loan_number*, *account_number* and *customer_name* as shown in table 2.6 and table 2.7.

Table 2.6 The Depositor Relation

customer_name	*account_number*
Jay	A-101
Shyam	A-215
Hayes	A-102
Turner	A-305
Jay	A-201
Jones	A-217
Lindsay	A-222

Table 2.7 The Borrower Relation

customer_name	*loan_number*
Jay	L-17
Shyam	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Shyam	L-11
Williams	L-17
Adams	L-16

Query for finding names of all customers who have loan in the bank can be given as

$$\Pi_{customer_name}\ (borrower)$$

Similarly, query to find name of all customers who have account in the bank can be given as

$$\Pi_{customer_name}\ (depositor)$$

Now, for finding the required result i.e., the customer name who has either an account or a loan at the bank, we have to union above two queries. Union operation is denoted by symbol ∪.

Expression needed is given as

$$\Pi_{customer_name}\ (borrower) \cup \Pi_{customer_name}(depositor)$$

Table 2.8 Name of all Customers who have either an Account or a Loan in Bank

customer_name
Jay
Shyam
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Adams

The result of this query can be given by table 2.8.

(iv) The Set Difference Operation – The set difference operation is denoted by '–'. It allows us to find tuples that are in one relation but are not in another. The expression r – s produces a relation containing those tuples in r

Asim Verma

but not in S. The set difference operator is a binary operator. For example, we can find all customers of the bank who have an account but not a loan by writing

$\Pi_{customer_name}$ (depositor) – $\Pi_{customer_name}$ (borrower). The result relation of this query can be shown by table 2.9.

Table 2.9 Customer with an Account but no Loan

customer_name
Johnson
Turner
Lindsay

So, by above discussion we have –

Notation : R – S
Purpose : To produce single relation removing the common tuples among R and S.
Input : Two relations R and S with identical schema
Output : Relation that has same schema as R and S.

(v) The Cartesian Product Operation – The cartesian product operation is denoted by a cross (×) symbol. It allows us to combine information from any two relations. We write cartesian product of two relations R_1 and R_2 as $R_1 \times R_2$. The cartesian product of any two relations R_1 (of degree m) and R_2 (of degree n) yields a relation $R_1 \times R_2$ of degree m + n. This product relation has all the attributes that are present in relations R_1 and R_2 and the tuples in $R_1 \times R_2$ are also possible combinations of tuples from R_1 and R_2. So, if cardinality of R_1 is x and cardinality of R_2 is y, then the cardinality of $R_1 \times R_2$ is xy. For example, let we want to find the names of all customers who have a loan at the perryridge branch. We need the information in both the *loan* relation and the *borrower* relation to do so. If we write

$$\sigma_{branch_name = \text{"Perryridge"}} (borrower \times loan)$$

then the result is shown in table 2.10.

Table 2.10 Result of $\sigma_{branch_name = \text{"Perryridge"}}$ (borrower × loan)

customer_name	*loan_number*	*branch_name*	*loan_number*	*amount*
Jay	L-17	Perryridge	L-15	1500
Jay	L-17	Perryridge	L-16	1300
Shyam	L-23	Perryridge	L-15	1500
Shyam	L-23	Perryridge	L-16	1300
Hayes	L-15	Perryridge	L-15	1500
Hayes	L-15	Perryridge	L-16	1300
Jackson	L-14	Perryridge	L-15	1500
Jackson	L-14	Perryridge	L-16	1300
Curry	L-93	Perryridge	L-15	1500
Curry	L-93	Perryridge	L-16	1300
Shyam	L-11	Perryridge	L-15	1500
Shyam	L-11	Perryridge	L-16	1300
Williams	L-17	Perryridge	L-15	1500
Williams	L-17	Perryridge	L-16	1300
Adams	L-16	Perryridge	L-15	1500
Adams	L-16	Perryridge	L-16	1300

But the above query executes only when cartesian-product results first, followed by restriction operation. The result of cartesian product of borrower and loan can be given by table 2.11.

Table 2.11 Result of borrower × loan

customer_name	*borrower loan_number*	*branch_name*	*loan loan_number*	*amount*
Jay	L-17	Downtown	L-17	1000
Jay	L-17	Redwood	L-23	2000
Jay	L-17	Perryridge	L-15	1500
Jay	L-17	Downtown	L-14	1500
Jay	L-17	Mianus	L-93	500
Jay	L-17	Round Hill	L-11	900
Jay	L-17	Perryridge	L-16	1300
Shyam	L-23	Downtown	L-17	1000
Shyam	L-23	Redwood	L-23	2000
Shyam	L-23	Perryridge	L-15	1500
Shyam	L-23	Downtown	L-14	1500
Shyam	L-23	Mianus	L-93	500
Shyam	L-23	Round Hill	L-11	900
Shyam	L-23	Perryridge	L-16	1300
Hayes	L-15	Downtown	L-17	1000
Hayes	L-15	Redwood	L-23	2000
Hayes	L-15	Perryridge	L-15	1500
Hayes	L-15	Downtown	L-14	1500
Hayes	L-15	Mianus	L-93	500
Hayes	L-15	Round Hill	L-11	900
Hayes	L-15	Perryridge	L-16	1300
⋮	⋮	⋮	⋮	⋮
Williams	L-17	Downtown	L-17	1000
Williams	L-17	Redwood	L-23	2000
Williams	L-17	Perryridge	L-15	1500
Williams	L-17	Downtown	L-14	1500
Williams	L-17	Mianus	L-93	500
Williams	L-17	Round Hill	L-11	900
Williams	L-17	Perryridge	L-16	1300
Adams	L-16	Downtown	L-17	1000
Adams	L-16	Redwood	L-23	2000
Adams	L-16	Perryridge	L-15	1500
Adams	L-16	Downtown	L-14	1500
Adams	L-16	Mianus	L-93	500
Adams	L-16	Round Hill	L-11	900
Adams	L-16	Perryridge	L-16	1300

MSInventek

So, cartesian product can be given as

Notation : $R_1 \times R_2$
Purpose : Pairs rows from two tables
Input : Two relations R_1 and R_2
Output : For each row in R_1 and each row in R_2, output a row R_1R_2; the output relations has the attribute of R_1 and the attributes of R_2. Its a binary operation.

(vi) The Rename Operation – The rename operation is used to give the name to the results of relational algebra expressions. It is denoted by the lowercase Greek letter rho (ρ). Given a relational algebra expression E, the expression

$$\rho_x (E)$$

returns the result of expression E under the name x.

A relation R by itself is a relational algebra expression. The rename operation can also be used to rename a relation R to get the same relation under a new name.

Therefore, another form of the rename operation is as follows. Assume that a relational algebra expression E has arity n. Then, the expression

$$\rho_x (A_1, A_2, ..., A_n) (E)$$

returns the result of expression E under the name x, and with the renamed attributes $A_1, A_2, ..., A_n$.

For example, we consider a query "find the largest account balance in the bank". For this purpose, we first compute a temporary relation consisting of those balances that are not the largest, and then to take the set difference between the relation $\Pi_{balance}$ (account) and the temporary relation just computed, to obtain the result. To compute the temporary relation, we need to compare the values of all account balances. We can do this by computing the cartesian product account × account and forming a selection to compare the values of any two balances appearing in one tuple. First, we need to devise a mechanism to distinguish between two balance attributes. We shall use the rename one reference to the account relation, thus, we can reference the relation twice without ambiguity. The temporary relation that consists of the balances, that are not the largest, can now be written as

$$\Pi_{account.balance} (\sigma_{account.balance < d.balance} (account \times \rho d (account)))$$

This expression gives those balances in the account relation for which a larger balance appears, somewhere in the account relation (renamed as d). The result contains all balances except the largest one. This relation is shown in Table 2.12.

Table 2.12 Result of Subexpression

$\Pi_{account.\ balance}$ $(\sigma_{account.\ balance < d.balance}$ **(account × ρd (account)))**

balance
500
700
400
350
750

The query to find the largest account balance in the bank can be written as follows –

$$\Pi_{balance} (account) - \Pi_{account.balance} (\sigma_{account.balance < d.balance} (account \times \rho d (account)))$$

Table 2.13 shows the result of this query.

Table 2.13 Largest Account Balance in the Bank

balance
900

So, by above discussion we can say that, for a rename operation

Notation : $\rho_s(R)$ or $\rho_{(A_1,A_2,..,A_m)} (R)$
Purpose : Rename a relation and/or this columns
Input : Relation R
Output : Relation S, with tuples of R under same or different named attributes.

Q.20. Give the usage of rename operation with an example. ***(R.G.P.V., Dec. 2014)***

Ans. Refer to Q.19.

Q.21. What is relational algebra ? Enlist and explain the fundamental operations of relational algebra. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.17 and Q.19.

Q.22. Explain set-intersection operation in relational algebra.

Ans. It returns a relation having all tuples that appear in both of two specified relations. Set-intersection operation is denoted by symbol '∩'.

For example, if we wish to find all customers who have both a loan and an account. Using set-intersection, we can write

$$\Pi_{customer_name} (borrower) \cap \Pi_{customer_name} (depositor)$$

Table 2.14 Customers with Both an Account and a Loan at the Bank

customer_name
Hayes
Jay
Shyam

The result relation for this query is shown in table 2.14.

Q.23. What is union compatibility ? Why do the union, intersection and set-difference operations require that the relations on which they are applied are union compatible ? ***(R.G.P.V., Nov./Dec. 2007)***

Or

What is union compatibility ? What are the relational algebra operators that require the relations on which they are applied be union compatible ? ***(R.G.P.V., June 2014, 2015)***

Uttarakhand Technical University

Or

What are the conditions to be fulfilled for two relations to be involved in a union operation ? Why do the union, intersection and set difference operation require the operand relations to be Union compatible ?

(R.G.P.V., June 2011)

Ans. Several set theoretic operations are used to merge the elements of two sets in various ways, including union, intersection, and set difference. These are binary operations; that is, each is applied to two sets. When these operations are adapted to relational databases, the two relations on which any of the above three operations are applied must have the same type of tuples. This condition is called ***union compatibility***. Two relations R (A_1, A_2,...., A_n) and S (B_1, B_2, ..., B_n) are said to be union compatible if they have the same degree n, and if dom (A_i) = dom (B_i) for $1 \le i \le n$. It means that the two relations have the same number of attributes and that each pair of corresponding attributes have the same domain.

The three operations union, intersection, and set difference can be defined on two union compatible relations R and S as follows –

(i) Union – The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

(ii) Intersection – The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S.

(iii) Set Difference – The result of this operation, denoted by R – S, is a relation that includes all tuples that are in R but not in S.

Q.24. Explain division operation in relational algebra.

Or

Explain the use of division relation operator with example.

(R.G.P.V., Dec. 2008)

Ans. The division operation is useful for a special kind of queries that include the phrase "for all". It is denoted by ÷.

It takes two unary relations and one binary relation and returns a relation containing all tuples from one unary relation that appear in the binary relation matched with all tuples in the other unary relation.

For example, suppose we wish to find all customers who have an account at all the branches located in Brooklyn. We can obtain all the branches located in Brooklyn by the expression –

$$R_1 = \Pi_{branch_name} (\sigma_{branch_city = \text{"Brooklyn"}} (branch))$$

The result of this relation is given in table 2.15.

Table 2.15 Result of $\Pi_{branch_name} (\sigma_{branch_city = \text{"Brooklyn"}} (branch))$

branch_name
Brighton
Downtown

Now, the second step is to find all (customer_name, branch_name) pairs for which the customers have an account at a branch by writing –

$$R_2 = \Pi_{customer_name,\ branch_name} (depositor \bowtie account)$$

Table 2.16 shows the result of this relation.

Table 2.16 Result of $\Pi_{customer_name,\ branch_name}$ (depositor ⋈ account)

customer_name	*branch_name*
Jay	Downtown
Shyam	Mianus
Hayes	Perryridge
Turner	Round Hill
Williams	Perryridge
Lindsay	Redwood
Johnson	Brighton
Jay	Brighton

Now, at last step we need to find customers who appear in R_2 with every branch name in R_1. The operation that provides exactly those customers is the divide operation. We formulate the query by writing –

$$\Pi_{customer_name,\ branch_name} (depositor \bowtie account) \div \Pi_{branch_name} (\sigma_{branch_city = \text{"Brooklyn"}} (branch))$$

The result of this expression is a relation that has the schema (customer_name) and that contains the tuple Jay.

Q.25. Explain natural join operation in relational algebra.

Or

Discuss where natural join operation is used in relational algebra.

(R.G.P.V., June 2016)

Ans. The natural join is a binary operation that allows us to combine certain selections and a cartesian product into one operation. It is denoted by the join symbol ⋈. The natural join operation forms a cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

Natural join is not as difficult as it looks. For example, "to find the names of all customers who have a loan at the bank and find the amount of the loan." We can express this query by following expression –

$$\Pi_{customer_name,\ loan_number,\ amount} (borrower \bowtie loan)$$

Msinventer

Since, the schemas for *borrower* and *loan* have the attribute *loan_number* in common, the natural-join operation considers only pairs of tuples that have same value on *loan_number*. It combines each such pair of tuples into a single tuple on the union of the two schemas i.e., customer_name, branch_name, loan_number, amount. After performing the projection, the resultant relation can be shown by table 2.17.

Table 2.17 Result of $\Pi_{customer_name, loan_number, amount}$ (borrower ⋈ loan)

customer_name	*loan_number*	*amount*
Jay	L-17	1000
Shyam	L-23	2000
Hayes	L-15	1500
Jackson	L-14	1500
Curry	L-93	500
Shyam	L-11	900
Williams	L-17	1000
Adams	L-16	1300

Now, the natural join can be formally defined as follows – If r (R) and s (S) are two relations, then the natural join of r and s, denoted by r⋈s, is a relation on schema R ∪ S defined as follows –

$$r \bowtie s = \pi_{R \cup S}(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots. \wedge r.A_n = s.A_n} r \times s)$$

where $R \cap S = \{A_1, A_2, \ldots, A_n\}$.

Q.26. With an example explain clearly JOIN and UNION operations in relational algebra. Bring out the difference between natural JOIN and OUTER JOIN. ***(R.G.P.V., June 2017)***

Ans. **JOIN and UNION** – Refer to the ans. of Q.25 and Q.19 (iii).

Difference between NATURAL JOIN and OUTER JOIN are as follows –

S.No.	*Natural Join*	*Outer Join*
(i)	Natural join is cartesian product on certain selection forcing equality on those attributes that appears in both tables or relations.	Outer join is cartesian product on certain selection without forcing any type of equality on attribute.
(ii)	Data loss may occur in natural join.	No loss of data in full outer join.
(iii)	Result does not contains any NULL value.	Mostly, result contains NULL values.

Q.27. Differentiate between inner join and outer join. ***(R.G.P.V., Dec. 2015)***

Ans. The inner join is the traditional join in which only rows that meet a given criteria are selected. The join criteria can be an equality condition (natural join or equijoin) or an equality condition (theta join). An outer join returns not only the matching rows, but also the rows with unmatched attribute values for one table or both tables to be joined.

Q.28. Explain assignment operation in relational algebra.

Ans. The assignment operation is denoted by ← sign. It works like assignment in a programming language. The evaluation of an assignment does not result in any relation being displayed to the user. The result of the expression to the right of the ← is assigned to the relation variable on the left of the ←. This relation variable may be used in subsequent expressions.

In the assignment operation, a query can be written as a sequential program consisting of a series of assignment followed by an expression whose value is displayed as the result of query. For relational algebra queries, assignment must be made to a temporary relation variable. Assignments to permanent relations constitute a database modification.

For example, if bank manager wants to know the customer_name who have account, then the relational expression would be –

$$result \leftarrow \Pi_{customer_name}(depositor)$$

The degree and cardinality of the resultant relation will depend on the expression on RHS.

Q.29. Discuss the different relational algebra operations. ***(R.G.P.V., June 2017)***

Ans. Refer to Q.19, Q.22, Q.24, Q.25 and Q.28.

Q.30. What are the relational algebra operations (other than set operations) developed specifically for a relational database ? ***(R.G.P.V., June 2010)***

Ans. Refer to Q.19, Q.24, Q.25 and Q.28.

Q.31. Explain JOIN. Differentiate between join and cartesian product. Give various join types and condition. ***(R.G.P.V., June 2007)***

Ans. The JOIN operation, denoted by ⋈, is used to combine related tuples from two relations into single tuples. To illustrate join, suppose that we want to retrieve the name of the manager of each department. To get the manager's name, we need to combine each department tuple with the employee tuple whose SSN value matches the MGRSSN value in the department tuple. This is done by using the join operation, and the projecting the result over the necessary attributes as follows –

$$DEPT_MGR \leftarrow DEPARTMENT \bowtie_{MGRSSN = SSN} EMPLOYEE$$

$$RESULT \leftarrow \Pi_{DNAME, LNAME, FNAME}(DEPT_MGR)$$

The first operation is shown in fig. 2.3.

DEPT_MGR	DNAME	DNUMBER	MGRSSN	•••	FNAME	MINIT	LNAME	SSN	•••
	Research	5	333445555	•••	Franklin	T	Wong	333445555	•••
	Administration	4	987654321	•••	Jennifer	S	Wallace	987654321	•••
	Headquarters	1	888665555	•••	James	E	Borg	888665555	•••

Fig. 2.3 Illustrating the JOIN Operation

The general form of JOIN operation on two relations $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_m)$ is

$$R \bowtie_{<join\ operation>} S$$

The result of the JOIN is a relation Q with n + m attributes $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$ in that order; Q has one tuple for each combination of tuples – one from R and one from S – whenever the combination satisfies the join condition. This is the main difference between CARTESIAN PRODUCT and JOIN – in JOIN, only combinations of tuples satisfying the join operation appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result. The join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples. Each tuple combination for which the join condition evaluates to true is included in the resulting relation Q as a single combined tuple.

A general join condition is of the form

<condition>AND<condition>AND....AND<condition>

where each condition is of the form $A_i\ \theta\ B_j$, A_i is an attribute of R, B_j is an attribute of S, A_i and B_j have the same domain, and θ(theta) is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$. A JOIN operation with such a general join condition is called a **THETA JOIN.** Tuples whose join attributes are null do not appear in the result.

The most common JOIN includes join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an EQUIJOIN. Note that in the result of an EQUIJOIN there have one or more pairs of attributes that have identical values in every tuple of DEPT_MGR because of the equality join condition specified on these two attributes. Because one of each pair of attributes with identical values is superfluous, a new operation called NATURAL JOIN, denoted by *, was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition. The NATURAL JOIN requires that the two join attributes have the same name in both relations. If this is not the case, a renaming operation is applied first.

Q.32. Discuss the selection, projection and join operator of relation algebra with a suitable example. ***(R.G.P.V., Dec. 2011)***

Ans. **Selection Operation** – Refer to Q.19 (i).

Projection Operation – Refer to Q.19 (ii).

Join Operator – Refer to Q.31.

Q.33. Explain theta join with an example.

Or

What are the various types of inner join operations ? Why theta join is required ? Explain by giving proper examples. (R.G.P.V., June 2006, Dec. 2009)

Or

Discuss the various types of inner join operations. Why is theta join required ? ***(R.G.P.V., June 2010)***

Ans. The theta join operation is an extension to the natural-join operation. This operation is intended to join two relations together on the basis of some comparison operator other than equality. Let relations A and B satisfy the requirements for cartesian product i.e., they have no attribute names in common. Let A have an attribute X and let B have an attribute Y, and let X, Y, and θ satisfy the requirements for restriction. Then, the θ join of relation A on attribute X with relation B on attribute Y is defined to be the result of evaluating the expression.

(A TIMES B) WHERE X θ Y

In other words, it is a relation with the same heading as the cartesian product of A and B, and with a body consisting of the set of all tuples t such that t appears in that cartesian product and the condition "X θ Y" evaluates to true for that tuple t.

If θ is ">", the θ-join is called the ***greater than join***. For example, suppose we wish to compute the greater-than join of relation S on city with P on city as shown in table 2.18 then the relational expression is as follows –

((S RENAME CITY AS SCITY) TIMES
(P RENAME CITY AS PCITY))
WHERE SCITY > PCITY

The result of this expression as shown in table 2.18.

Table 2.18 Greater-than Join of Suppliers and Parts on Cities

S#	SNAME	STATUS	SCITY	P#	PNAME	COLOR	WEIGHT	PCITY
S_2	Jones	10	Paris	P_1	Nut	Red	12.0	London
S_2	Jones	10	Paris	P_4	Screw	Red	14.0	London
S_2	Jones	10	Paris	P_6	Cog	Red	19.0	London
S_3	Blake	30	Paris	P_1	Nut	Red	12.0	London
S_3	Blake	30	Paris	P_4	Screw	Red	14.0	London
S_3	Blake	30	Paris	P_6	Cog	Red	19.0	London

If θ is " = ", the θ-join is called an ***equijoins.*** It follows from the definition that the result of an equijoin must include two attributes with the property that the values of those two attributes are equal in every tuple in the relation. If one of those two attributes is projected away and the other renamed appropriately, the result is the natural join.

Similarly, if θ is "<", the θ-join is called the ***less than join.*** For example, to compute the less than join of relations S on city with P on city as shown in table 2.19, the relational expression is as follows –

((S RENAME CITY AS SCITY) TIMES
(P RENAME CITY AS PCITY))
WHERE SCITY < PCITY

Table 2.19 The Suppliers and Parts Database

S

S#	SNAME	STATUS	CITY
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CI
P_1	Nut	Red	12.0	Lond
P_2	Bolt	Green	17.0	Pari
P_3	Screw	Blue	17.0	Rom
P_4	Screw	Red	14.0	Lond
P_5	Cam	Blue	12.0	Pari
P_6	Cog	Red	19.0	Lond

The result of this expression is shown in table 2.20.

Table 2.20 Less-than Join of Suppliers and Parts on Cities

S#	SNAME	STATUS	SCITY	P#	PNAME	COLOR	WEIGHT	PCI
S_1	Smith	20	London	P_2	Bolt	Green	17.0	Pari
S_1	Smith	20	London	P_3	Screw	Blue	17.0	Rom
S_1	Smith	20	London	P_5	Cam	Blue	12.0	Pari
S_2	Jones	10	Paris	P_3	Screw	Blue	17.0	Rom
S_3	Blake	30	Paris	P_3	Screw	Blue	17.0	Rom
S_4	Clark	20	Paris	P_3	Screw	Blue	17.0	Rom
S_5	Adams	30	London	P_2	Bolt	Green	17.0	Pari
S_5	Adams	30	London	P_3	Screw	Blue	17.0	Rom
S_5	Adams	30	London	P_5	Cam	Blue	12.0	Pari

Q.34. Discuss the following extended relational algebra operations
(i) Generalized projection
(ii) Outer join
(iii) Aggregate functions.

Ans. (i) Generalized Projection – The generalized projection is t extension of the projection operation. It allows to use arithmetic functions the projection list. The generalized projection operation can be written as

$$\Pi_{F_1, F_1, \ldots\ldots, F_n}(E)$$

where, E is any relational algebra expression, and F_1, F_2, ..., F_n are arithmet expressions involving constants and attributes in the schema of E. In a speci case, the arithmetic expression may be an attribute or a constant.

For example, suppose we have a relation credit-into as shown in table 2.2 which lists the credit limit and expenses so far (i.e., the credit_balance on th account).

Table 2.21 The credit_info Relation

customer_name	*limit*	*credit_balance*
Pavan	2000	1700
Rakesh	1500	1400
Rajesh	6000	2000
Kamal	2000	1000

If we want to know that how much each customer can spend, we can write the expression as follows –

$$\Pi_{customer_name,\ limit - credit_balance}(credit_info)$$

Here, the attribute resulting from the expression limit – credit_balance does not have a name. Now, the rename operation can be used to give it a name. Therefore, the expression can now be written as –

$$\Pi_{customer_name,\ (limit - credit_balance)\ as\ credit_available}(credit_info)$$

Table 2.22 shows the result of applying this expression to the relation in table 2.21.

Table 2.22

customer_name	*credit_available*
Pavan	300
Rakesh	100
Rajesh	4000
Kamal	1000

(ii) Outer Join – The outer join operation is an extension of the join operation to deal with missing information. Suppose that there are two relations with the following schemas containing data on full-time employees –

employee (emloyee_name, street, city)

ft_works (employee_name, branch_name, salary)

Table 2.23 shows two relations employee and ft_works.

Table 2.23 The employee and ft_works Relations

employee_name	street	city
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
William		Seattle

employee_name	branch_name	salary
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
William	Redmond	1500

If we want to generate a single relation with all the information about full_time employee. The approach is to use the natural join operation as follows –

employee ⋈ ft_works

The result of this operation is shown in table 2.24.

Table 2.24 The Result of employee ⋈ ft_works

employee_name	street	city	branch_name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
William	Seaview	Seattle	Redmond	1500

Msinvented

Here, we have lost the street and city information about smith, since t tuple describing smith is absent from the ft_works relation. Similarly, t branch_name and salary information about Gates have been lost, since t tuple describing Gates is absent from the employee relation.

This loss of information can be avoided using outer join operation. The are three forms of outer join operation – left outer join, right outer join, and f outer join. All three forms of outer join compute the join, and add extra tupl to the result of the join.

(a) Left Outer Join – It is denoted by ⟕ symbol. It takes a tuples in the left relation that did not match with any tuple in the right relatio pads the tuples with null values for all other attributes from the right relatio and adds them to the result of the natural join. Table 2.25 shows the result the expression employee ⟕ ft_works.

Table 2.25 Result of employee ⟕ ft_works

employee_name	street	city	branch_name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
William	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	Null	Null

(b) Right Outer Join – It is denoted by ⟖ symbol. It pad tuples from the right relation that did not match from the left relation with nul values and adds them to the result of the natural join.

Table 2.26 shows the result of the expression employee ⟖ ft_works.

Table 2.26 Result of employee ⟖ ft_works

employee_name	street	city	branch_name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
William	Seaview	Seattle	Redmond	1500
Gates	Null	Null	Redmond	5300

(c) Full Outer Join – The full outer join is denoted by ⟗ symbol performs functions of both left and right outer joins. This means tha it pads tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adds them to the result of the join. Table 2.27 shows the result of expression employee ⟗ ft_works.

Table 2.27 Result of employee ⟗ ft_works

employee_name	street	city	branch_name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
William	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	Null	Null
Gates	Null	Null	Redmond	5300

(iii) Aggregate Functions – Aggregate functions allow a user to perform computations on the values of the relation's attributes. These are the functions that take a collection of values as input and return a single value as output. These are basically mathematical in nature. The aggregate functions with their purpose are listed below –

(a) Sum – Takes a collection of values of attributes and returns the sum of all of them.

(b) Avg – Takes a collection of values of attribute and returns the average of them.

(c) Min – Determines and returns minimum among the set of values of attributes.

(d) Max – Determines and returns maximum value among the set of values of attributes.

(e) Count – Counts and returns the number of values in the attribute.

Q.35. Explain the following relational algebra operations –
(i) Natural Join (ii) Assignment
(iii) Generalized projection (iv) Set intersection.
(R.G.P.V., June 2014)

Ans. Refer to Q.25, Q.28, Q.34 (i) and Q.22.

Q.36. Explain join, natural join, outer join, full outer join, left outer join and right outer join with example. (R.G.P.V., Nov./Dec. 2007, Dec. 2016)

Or

Explain/discuss various types of joins with examples.
(R.G.P.V., June 2013, May 2018)

Ans. Refer to Q.31, Q.25, Q.34 (ii).

Q.37. What are views ? How views are created ? Can views be updated ? What are the constraints ? Give examples.

Or

What is the significance of using the views ? Write a command to create the view. (R.G.P.V., Dec. 2010, June 2012)

Or

What are views ? Explain what are the major problems with processing update operations expressed in terms of views. (R.G.P.V., June 2013)

Ans. A view is an abstract model of a portion of the conceptual database. Many, but not all database management systems provide a facility to declare views called subschema data definition language and the facility to express

queries and operations on the views that would be called a subschema data manipulation language. For example, an airline system provides us a computerized reservation service through which we can view only data and a collection of programs that deals with flights and passengers. But through these programs, we can not know about personnel files or the assignments of pilots to flights as we do not need for that. But at the same time, the dispatcher may need to know about flights, aircrafts and aspects of personnel files (e.g., which pilots are qualified to fly a 470), but they do not need to know about personnel salaries or the passengers booked on a flight. So, we can say that, there can be two different views of the database. One for the reservation department and another for the dispatcher's office. In other words, view provides abstraction.

To define a view, we must give the view a name and must state the query that computes the view. The form of the create view command is

create view viewname as <query expression>
where <query expression>

is any legal query expression. The value of a given view at a given time is whatever results from evaluating a certain relational expression at that time.

For example, the statement –

create view TOPEMP as (emp where salary > 35K)
{emp#, ename, salary };

might be used to define a view called TOPEMP.

Whenever this statement is executed, the relational expression following as – the view definition expression – is not evaluated, but merely "remembered" by the system in some way (actually by saving it in the catalog, under the specified name, TOPEMP). User can operate on the view, just as if it were a base relation. View might be thought as virtual relation. On the contrary, view is just a window into the underlying base relation. As a consequence, any changes to that underlying relation will be automatically and instantaneously visible through that window. Likewise, changes to view will automatically and instantaneously be applied to base relation. When a tuple in a view can be mapped to a tuple in a base relation, the update may be made. However, when the tuple in a view does not map to a single tuple, the update operation may not be determined unambiguously.

A tuple in a view can be theoretically updated under the following constraints –

(i) Any update operation through a view requires that the user has appropriate authorization.

(ii) If the definition of view involves a simple query on a single base relation and includes the primary key, the following update operations are possible – a new tuple can be inserted into the database through a view, an existing tuple can be deleted through a view, and the value of a nonprime attribute can be modified. The simple query provision rules out the possibility that the attributes in the view are derived using an aggregate function or a nonreversible operation. For example, if the view uses the value of weight in pounds and the value in the base relation stored in grams, the view attribute weight is obtained by dividing the base value by 453.6 and the stored value, inserted through the view, is multiplied by the same amount.

(iii) The insertion of a new record using a view requires that the primary attributes are included in the view, and the values for these are specified for the inserting (i.e., they are not null).

Although views are a useful tool for queries, they cause serious problems if we express updates, insertions, or deletions with them. The difficulty is that a modification to the database expressed in terms of a view must be translated to a modification to the actual relations is the logical model of the database.

For example, consider a clerk who want to see all loan data in the loan relation, except loan-amount. Let loan-branch be the view given to the clerk. This view is defined as

Create view loan_branch as
select loan_number, branch_name from loan

Since we allow a view name to appear wherever a relation name is allowed, the clerk can write –

insert into loan_branch values ('L-307', 'Perryridge')

This insertion must be represented by an insertion into the relation loan, since loan is the actual relation from which the database system constructs the view loan_branch. However, to insert a tuple into loan, we must have some value for amount. There are two approaches to deal with this insertion –

(i) Reject the insertion, and return a error message to the user.

(ii) Insert a tuple (L-37, "Perryridge", null) into the loan relation.

Another problem with modification of the database through views occurs with a view such as –

create view loan-info as
select customer_name, amount
from borrower, loan where borrower.loan_number = loan.loan_number

This view lists the loan amount for each loan that any customer of the bank has consider the following insertion through this view –

loan_info ← loan_info v {("Johnson", 1900)}

The only possible method of inserting tuples into the borrower and loan relations is to insert ("Johnson", null) into borrower and (null, null, 1900) into loan. Then, the obtained relations are as shown in table 2.28.

Table 2.28 Tuples Inserted into loan and borrower

loan_number	branch_name	amount
L-11	Roundhill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500
null	null	1100

customer_name	loan_number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
William	L-17
Johnson	null

However, this update does not have the desired effect, since the view relation loan_info still does not include the tuple ("Johnson", 1900). Thus, there is no way to update the relations borrower and loan by using nulls to get the desired update on loan_info. Because of these problems, modifications are generally not permitted on view relations, except in limited cases.

In summary, the conditions under which most DBMS determine whether an update is allowed through a view, are as follows –

(i) Updates are allowed through a view defined using a simple query involving a single base relation and containing either the primary key or a candidate key of the base relation.

(ii) Updates are not allowed through views involving multiple relations.

(iii) Updates are not allowed through views involving aggregate or grouping operations.

Advantages –

(i) Views provide atomic security for hidden data.

(ii) Views provide a shorthand or "macro" capability.

(iii) Views allow the same data to be seen by different users in different ways at the same time.

(iv) Views can provide logical data independence.

Q.38. Write short note on relational calculus. ***(R.G.P.V., June 2015)***

Ans. Relational calculus is a formal query language where we write one ***declarative*** expression to specify a retrieval request and hence there is no description of how to evaluate a query; a calculus expression specifies what is to be retrieved rather than how to retrieve it. Therefore, the relational calculus is considered to be a ***nonprocedural*** language. This differs from relational algebra, where we must write a sequence of operations to specify a retrieval request. Hence, it is considered as a ***procedural*** way of stating a query. It is possible to rest algebra operations to form a single expression. However, a certain order among the operations is always explicitly specified in a relational algebra expression. This order also influences the strategy for evaluating the query. Any retrieval that can be specified in the algebra can also be specified in the relational calculus and vice versa. In other words, the expressive power of the two languages is identical.

The fact is, the algebra and the calculus are logically equivalent. For every algebraic expression there is an equivalent calculus one, for every calculus expression there is an equivalent algebraic one. There is a one-to-one correspondence between the two. Thus, the difference between them is really just a difference of style – the calculus is arguably closer to natural language, the algebra is like a programming language.

Relational calculus is based on a branch of mathematical logic called ***predicate*** calculus. The idea of using predicate calculus as the basis for a query language was originated by Kuhns. The concept of a relational calculus – i.e., an applied predicate specifically tailored to relational databases – was first proposed by Codd. A language explicitly based on that calculus called data sublanguage ALPHA was presented by Codd. ALPHA was never implemented, but a language called QUEL was implemented and had been a competitor to SQL.

Q.39. Differentiate between relational algebra and relational calculus. ***(R.G.P.V., June 2014)***

Ans. Refer to Q.17 and Q.38.

Q.40. What do you understand by relational query languages ?

Ans. A query language is a language in which a user requests information from the database. Query languages can be categorized as being either procedural or non-procedural. In procedural language, user instructs the system to perform a sequence of operations on the database to compute the desired result. In a non-procedural language, the user describes the information desired without giving a specific procedure for obtaining that information. The relational algebra is a procedural, whereas the tuple relational calculus and domain relational calculus are nonprocedural.

In relational query languages, we are concerned with queries. A complete data-manipulation language includes not only a query language, but also a language for database modification. Such language includes command to insert and delete tuples as well as commands to modify parts of existing tuples.

Q.41. Explain with respect to tuple calculus –
(i) Tuple variable ***(ii) Range relation***
(iii) Atom formula ***(iv) Expression***
(v) $\exists$ (Existential quantifier) and $\forall$ (universal quantifier).
(R.G.P.V., Nov./Dec. 2007)

Ans. The tuple relational calculus is a nonprocedural query language. It defined as the desired information without giving a specific procedure for

obtaining that information. It is based on specifying a number of ***tuple variables***. Each tuple variable usually ***ranges over*** a particular database relation, it's meant that the variable may take as its value any individual tuple from that relation. A query in the tuple relational calculus is expressed as

$$\{t \mid COND\ (t)\}$$

where t is a tuple variable and COND(t) is a conditional expression involving t. The result of such a query is the set of all tuples t that satisfy COND(t). For example, to find the employee whose salary is above $ 50,000, the tuple calculus expression is as follows –

$$\{ t \mid employee\ (t)\ and\ t.salary > 50000\}$$

The condition employee (t) specifies that the range relation of tuple variable t is employee. Each employee tuple t that satisfies the condition t.salary > 5000 is retrieved. If we want to select only some of the attributes – say the first and last names, then the query is written as follows –

$$\{t.fname, t.lname \mid employee\ (t)\ and\ t.salary > 50000\}$$

Informally, the following information is specified in a tuple calculus expression –

(i) For each tuple variable t, the range relation R of t. This value is specified by a condition of the form R(t).

(ii) A condition to select particular combinations of tuples. As tuple variables range over their respective range relations, the condition is evaluated for every possible combination of tuples to identify the selected combinations for which the condition evaluates to true.

(iii) A set of attributes to be retrieved i.e., the requested attributes. The values of these attributes are retrieved for each selected combination of tuples.

Expressions and Formulas in Tuple Relational Calculus – A general ***expression*** of the tuple relational calculus is of the form

$$\{t_1.A_1, t_2.A_2, ..., t_n.A_n \mid COND\ (t_1, t_2, ..., t_n, t_{n+1}, t_{n+2}, ..., t_{n+m})\}$$

where $t_1, t_2, ..., t_n, t_{n+1}, ..., t_{n+m}$ are tuple variables, each A_i is an attribute of the relation on which t_i ranges, and COND is a ***condition*** or ***formula*** of the tuple relational calculus. A formula consists of predicate calculus atoms, which can be one of the following –

(i) An atom of the form $R(t_i)$, where R is a relation name and t_i is a tuple variable. This atom identifies the range of the tuple variable t_i as the relation whose name is R.

(ii) An atom of the form $t_i.A\ op\ t_j.B$, where op is one of the comparison operators in the set $\{ =, >, \geq, <, \leq, \neq\}$, t_i and t_j are tuple variables, A is an attribute of the relation on which t_i ranges, and B is an attribute of the relation on which t_j ranges.

(iii) An atom of the form $t_i.A$ op c or c op $t_j.B$, where op is one of the comparisons operators in the set $\{ =, >, \geq, <, \leq, \neq\}$, t_i and t_j are tuple variables, A is an attribute of the relation on which t_i ranges, B is an attribute of the relation on which t_j ranges and c is a constant value.

Every atoms evaluates to either true or false for a specific combination of tuples. This called the ***truth value*** of an atom. For atoms of type1, if the tuple variable is assigned a tuple that is a member of the specified relation R, the atom is true; otherwise it is false. For atoms of type 2 and 3, if the tuple variables are assigned to tuples such that the values of the specified attributes of the tuples satisfy the condition, then the atom is true.

A ***formula*** or ***condition*** consists of one or more atoms connected through the logical operators ***and, or,*** and ***not,*** is defined recursively as follows –

(i) Every atom is a formula

(ii) If F_1 and F_2 are formulas, then so are (F_1 and F_2), (F_1 or F_2), not (F_1), and not (F_2). The truth values of these four formulas are derived from their component formulas F_1 and F_2 as follows –

(a) (F_1 and F_2) is true if both F_1 and F_2 are true, otherwise false.

(b) (F_1 or F_2) is false if both F_1 and F_2 are false; otherwise true.

(c) not (F_1) is true if F_1 is false, it is false if F_1 is true.

(d) not (F_2) is true if F_2 is false, it is false if F_2 is true.

The Existential (∃) and Universal (∀) Quantifiers – Two special symbols called quantifiers can appear in formulas; these are the universal quantifier (∀) and the existential quantifier (∃). A tuple variable t is ***bound*** if it is quantified. It means that it appears in an (∃t) or (∀t) clause; otherwise, it is ***free***. Formally, a tuple variable can be defined in a formula as free or bound according to the following rules –

(i) An occurrence of a tuple variable in a formula F that is an atom is free in F.

(ii) An occurrence of a tuple variable t is free or bound in a formula made up logical connectives – (F_1 and F_2), (F_1 or F_2), not (F_1), and not (F_2) – depending on whether it is free or bound in F_1 or F_2.

(iii) All free occurrences of a tuple variable t in F are bound in formula F' of the form F' = (∃t) (F) or F' = (∀t)F. The tuple variable is bound to the quantifier specified in F'.

Now, the rules for the definition of a formula are as follows –

(i) If F is a formula, the so is (∃t) (F), where t is tuple variable. The formula (∃t) (F) is true if the formula F evaluates to true for at least one tuple assigned to free occurrences of t in F; otherwise (∃t) (F) is false.

(ii) If F is a formula, then so is (∀t) (F), where t is a tuple variable. The formula (∀t) (F) is true if the formula F evaluates to true for every tuple assigned to free occurrences of t in F; otherwise (∀t) (F) is false.

MSU Verified

The (∃) quantifier is called an existential quantifier because a formula (∃t) (F) is true if there exists some tuple that makes F true. For the universal quantifier, (∀t) (F) is true if every possible tuple that can be assigned to free occurrences of t in F is substituted for t, and F is true for every such substitution. It is called the universal (or for all) quantifier because every tuple in the universe of tuples must make F true to make the quantified formula true.

Q.42. What do you mean by domain relational calculus ? Explain its operations in brief.

Or

Differentiate between tuple oriented and domain oriented relational calculus. ***(R.G.P.V., Dec. 2008)***

Ans. Another type of relational calculus is called the domain relational calculus or simply domain calculus.

The key difference between tuple calculus and domain calculus is that the range variables of the tuple calculus range over relations, while the range variables of the domain calculus range over domains.

An expression in the domain relational calculus is of the form

$$\{<x_1, x_2, ..., x_n> \mid P(x_1, x_2, ..., x_n)\}$$

where $x_1, x_2, ..., x_n$ are domain variables that range over domains (of attributes) and P is a condition or formula composed of atoms. An atom in the domain relational calculus can be one of the following forms –

(i) $<x_1, x_2, ..., x_n> \in r$, where r is a relation on n attributes and $x_1, x_2, ..., x_n$ are domain variables or domain constants.

(ii) $x \theta y$, where x and y are domain variables and θ is comparison operator in the set $\{=, >, \geq, <, \leq, \neq\}$.

(iii) $x \theta c$, where x is a domain variable θ is comparison operator in the set $\{=, >, \geq, <, \leq, \neq\}$, and c is a constant in the domain of the attribute for which x is a domain variable.

As in tuple calculus, atoms evaluate to either true or false for a specific set of values, called the truth values of the atoms. In case 1, if the domain variables are assigned values corresponding to a tuple of the specified relation r, then the atom is true. In cases 2 and 3, if the domain variables are assigned values that satisfy the condition, then the atom is true.

We can build up formulae from atoms by using the following rules –

(i) An atom is a formula.

(ii) If P_1 is a formula, then so are $\neg P_1$ and (P_1).

(iii) If P_1 and P_2 are formulae, then so are $P_1 \vee P_2$, $P_1 \wedge P_2$, and $P_1 \Rightarrow P_2$.

(iv) If P_1 is a formula in x, where x is a domain variable, then $\exists x (P_1(x))$ and $\forall x (P_1(x))$ are also formulae.

Q.43. Explain the following terms to show your understanding of their essential features –

(i) Relational calculus (ii) Tuple calculus (iii)Domain calculus. ***(R.G.P.V., June 2010)***

Ans. (i) Relational Calculus – Refer to Q.38.

(ii) Tuple Calculus – Refer to Q.41.

(iii) Domain Calculus – Refer to Q.42.

Q.44. Explain what is meant by repetition of information and inability to represent information. Explain why each of these properties may indicate a bad relational database design. ***(R.G.P.V., Dec. 2009)***

Ans. The undesirable properties that a bad database design may have are –

(i) Repetition of information

(ii) Inability to represent certain information.

Consider the relation, *lending*, which is defined over the relation schema.

Lending_schema = (branch_name, branch_city, assets, customer_name, loan_number, amount)

Fig. 2.4 shows an instance of the relation lending (lending_schema). A tuple t in the lending relation has the following intuitive meaning –

(i) t [*assets*] is the asset figure for the branch named t [*branch_name*]

(ii) t [*branch_city*] is the city in which the branch named t [*branch_name*] is located.

(iii) t [*loan_number*] is the number assignd to a loan made by the branch named t [*branch_name*] to the customer named t [*customer_name*].

(iv) t [amount] is the amount of the loan whose number is t [*loan_number*].

branch_name	branch_city	assets	customer_name	loan_number	amount
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horsneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500
Mianus	Horseneck	400000	Jones	L-93	500
Roud hill	Horseneck	8000000	Turner	L-11	900
Powal	Bennington	300000	Williams	L-29	1200
North town	Rye	3700000	Hayes	L-16	1300
Dowtown	Brooklyn	9000000	Johnson	L-18	2000
Perryridge	Horseneck	1700000	Glenn	L-25	2500
Brighton	Brooklyn	7100000	Brooks	L-10	2200

Fig. 2.4 Lending Relation

Assume that we wish to add a new loan to our database. Say that the loan is made by the Perryridge branch to Adams in the amount of $1500. Let the loan number be L-31. In our design, we need a tuple with values on all the attributes of ***Lending_schema***. So, we must repeat the asset and city data for the Perryridge branch, and must add the tuple

(Perryridge, Horseneck, 1700000, Adams, L-31, 1500)

to the lending relation. In general, the asset and city data for a branch mu appear once for each loan made by that branch.

The repetition of information required by the use of this design undesirable. Repeating information wastes space. Furthermore, it complicat updating the database. Suppose, for example, that the assets of the Perryrid branch change from 1700000 to 190000. Then, many tuples of the *lendi* relation need to be changed. Thus, updates are more costly. When we perfo the update in the database, we must ensure that every tuple pertaining to t perryridge branch is updated, or else our database will show two differe assets values for the perryridge branch.

That observation is central to understanding why this design is bad. W know that a bank branch is located in exactly one city. On the other hand, w know that a branch make many loans. In other words, the functional dependenc

branch_name → branch_city

holds on lending_schema, but we do not expect the functional dependenc branch_name → loan_number to hold. The fact that a branch is located in city and the fact that a branch makes a loan are independent. These facts a best represented in separate relations.

Another problem with the *Lending_schema* design is that we cannc represent directly the information concerning a branch (*branch_name branch_city, assets*) unless there exists at least one loan at the branch. Th problem is that tuples in the *lending* relation require values for *loan_numbe amount,* and *customer_name.*

One solution to this problem is to introduce *null values* to handle updates through views.

Q.45. Consider the following relational database –

Employee (EMP_Name, Street, City)
Works (EMP_Name, Company_Name, Salary)
Company (Company_Name, City)
Manages (EMP_Name, Manager_Name)

For each of the following queries, give an expression in the relational algebra –

(i) Find the names of all employees who work for Satyam.
(ii) Find the names and cities of residence who work for Satyam.
(iii) Find the names of all employees who live in the same city as the company for which they work.
(iv) Find the names of all employees who do not work for Satyam.
(v) Find the names of all employees who live in the same city and on the same street as do their managers.

(R.G.P.V., Dec. 2013)

Ans. (i) $\Pi_{EMP_Name}(\sigma_{Company_Name = 'Satyam'}(Works))$

(ii) $\Pi_{EMP_Name, City}(Employee \bowtie \pi_{City}(\sigma_{Company_Name='Satyam'}(Company))$

(iii) $\Pi_{EMP_Name}(\sigma_{Employee.City=Company.City}(Employee \times (Works \times Company))$

(iv) $\Pi_{EMP_Name}(Works) - \pi_{EMP_Name}(\sigma_{Company_Name='Satyam'}(Works))$

(v) $\Pi_{Employee.EMP_Name}(\sigma_{Employee.Street\backslash Managers.Street \wedge Employee.City=Managers.City}(Employee \times \rho_{Managers(Street, City)}(\Pi_{Street, City}(\sigma_{Employee.EMP_Name=Manages.Manager_Name}(Employee \times Manages)))))$

Q.46. Consider the following relations for a database that keeps track of business trips of sales persons in a sales office : Salesperson (Salespersonid, Name, Start-year, Dept-no) Trip (Salespersonid, from, to, Departure-date, Return-date, trip-id).

Expense (trip-id, Account No., Amount)

Specify the foreign keys for the above schema. Then specify the following queries in

Relational algebra –

(i) Give the details (all attributes of trip relation) for trip that exceeded 10,000/- in expenses.

(ii) Print the 'Salespersonid' and 'Name' of the salespersons who took trips to 'delhi'.

(iii) Print the total trip expenses incurred by the salesman with salespersonid = '504'.

(R.G.P.V., June 2017)

Ans. Foreign key : trip-id and salespersonid

(i) $\Pi_{Salespersonid, from, to, Departure-date, Return-date, Trip-id}(\sigma_{Amount > 10000}(Trip \bowtie Expense))$

(ii) $\Pi_{Salespersonid, Name}(\sigma_{to = 'delhi'}(Salesperson \bowtie Trip))$

(iii) $\Pi_{Amount}(\sigma_{Salespersonid = '504'}(Trip \bowtie Expense))$

Q.47. Let the following relational schema be given –

Employee (SSN, name, age, dno)
Salary (SSN, salary)
Works_on (Project#, SSN)
Project (Project#, project_name, location)

For each of the following queries give an expression in tuple relational calculus –

(i) Display the names of projects at "Delhi".
(ii) Find the project_name of employee whose salary is greater than 10000.
(iii) Retrieve the name and SSN of employees working on project #A100.

(R.G.P.V., Dec. 2013)

Ans. (i) {t.project_name|Project(t) and t.location = "Delhi"}

(ii) {t|∃S∈ salary ∃W∈ Works_on ∃P∈ Project (P.Project# = W.Project # ∧ W.SSN = S.SSN ∧ S.salary > 10000 ∧ t.project_name = P.project_name)}

(iii) {t|∃E∈ Employee ∃W∈ Works_on (E.SSN = W.SSN ∧ W.Project # = "A100" ∧ t.name = E.name ∧ t.SSN = E.SSN)

ENTITY-RELATIONSHIP MODEL – BASIC CONCEPTS, DESIGN PROCESS, CONSTRAINTS, KEYS, DESIGN ISSUES, E-R DIAGRAMS, WEAK ENTITY SETS, EXTENDED E-R FEATURES – GENERALIZATION, SPECIALIZATIONS AND AGGREGATION

Q.48. Define E-R diagram. Draw an E-R diagram for library management system. Assume relevant entities and attributes for given system (R.G.P.V., May 2018)

Ans. The Entity-Relationship (E-R) Model is a graphical representation of the database logic and includes a detailed description of all entities, relationships and constraints.

The E-R data model is based on a perception of a real world that consists of a set of basic objects called ***entities***, and of ***relationships*** among these objects. It was developed to facilitate database design by allowing the specification of an ***enterprise schema***, which represents the overall logical structure of a database.

E-R Diagram for Library Management System –

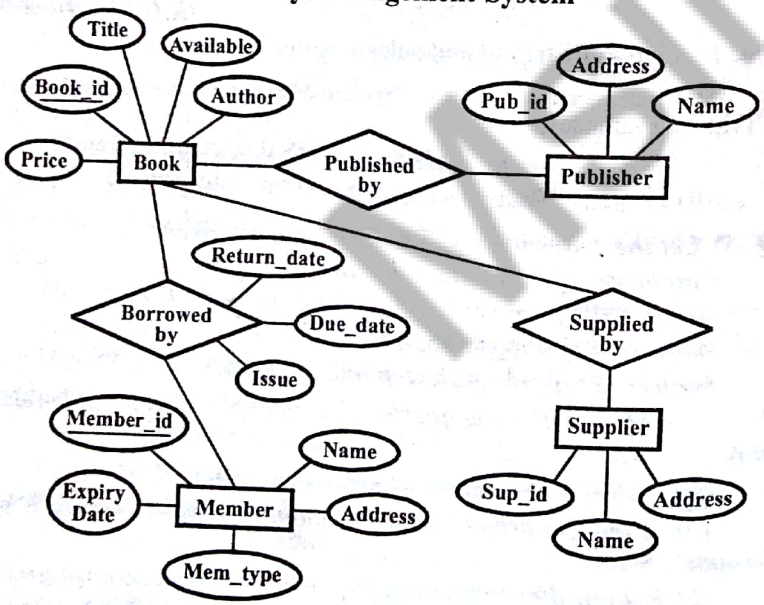


Fig. 2.5 E-R Diagram

Q.49. Develop an E-R diagram for library management system. (R.G.P.V., June 2012)

Or

Draw an E-R diagram for a library management system, make suitable assumptions. (R.G.P.V., Dec. 2013)

Sol. Refer to Q.48.

Q.50. Explain how the entity relationship (ER) model helped to produce a more structured relational database design environment. (R.G.P.V., Dec. 2011)

Ans. Complex design activities need conceptual simplicity to produce successful results. Even though the relational model was significantly improvable over the network and hierarchical models, it still lacked the features that would make it a dominant database design tool. Database designers mostly choose a graphical tool in which entities and their relationships are graphically represented because it is easier to understand structures graphically than to describe them in text. Thus, the entity relationship (ER) model is preferred as a widely accepted standard for data modeling.

The ER model was first introduced by Peter Chen in 1976. It was the graphical representation of entities and their relationships in a database structure that quickly become famous because it complemented the relational data model principles. The combination of the ERM and relational data model is used to provide the base for tightly structured database design. ER models can be generally represented in an entity relationship diagram (ERD) which uses graphical representations to model database components.

Q.51. What is a relationship type ? Explain the differences among a relationship instance, a relationship type and a relationship set. (R.G.P.V., June 2010)

Ans. A ***relationship set*** is a set of relationships of the same type. Formally, it is a mathematical relation on n ≥ 2 (possibly nondistinct) entity sets. If E_1, E_2, E_3,....,E_n are entity sets, then a relationship set R is a subset of

$$\{(e_1, e_2,.....,e_n)\mid e_1 \in E_1,\ e_2 \in E_2,\ .....,e_n \in E_n\}$$

where $(e_1, e_2,.....,e_n)$ is a relationship.

Let's there are two entity sets namely ***customer*** and ***loan.*** The relationship set ***borrower*** is defined to indicate the association between customers and the bank loans that the customers have. This association is shown in fig. 2.6.

In an E-R schema, a ***relationship instance*** represents an association between the named entities in the real-world enterprise that is being modeled. For example, the single ***customer*** entity Jackson, who has customer identifier 555-55-5555, and that ***loan*** entity L-14 participate in a relationship instance of ***borrower.*** In the real-world enterprise, this relationship instance represents

MEI Notes

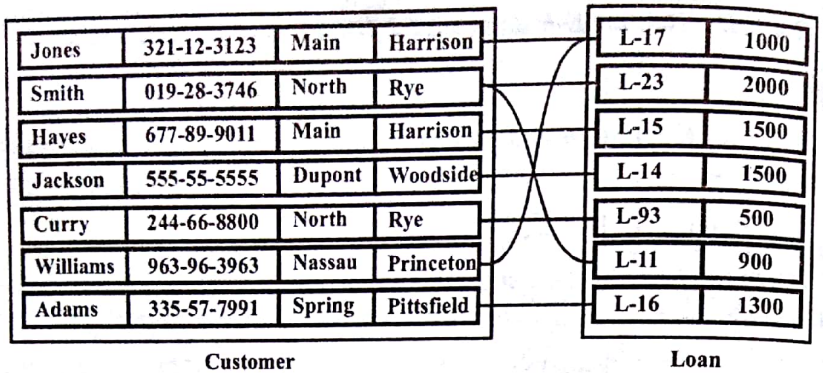


Fig. 2.6 Relationship Set Borrower

that, the person called Jackson who holds customer-id 555-55-5555 has taken the loan that is numbered L-14.

Relationship type indicates the association between entity types. e.g., consider the entity types, students and courses, having relationship "Completes". This is a many-to-many relationship, as each student can complete any number of courses, while a given course can be completed by any number of students. Relationship instance is an association between entity instances. Say, for example, Mr. Vinit completes the course of C++. Thus, entity instance is an association between entity instances, where each entity instance is the member of two participating entity types. In general, we can say that relationship types is a union of relationship instances of the participating entities of the given entity types.

Q.52. What is meant by mapping cardinalities ? For a binary relationship set what are the possible mapping cardinalities ? Explain with diagram. ***(R.G.P.V., June 2014)***

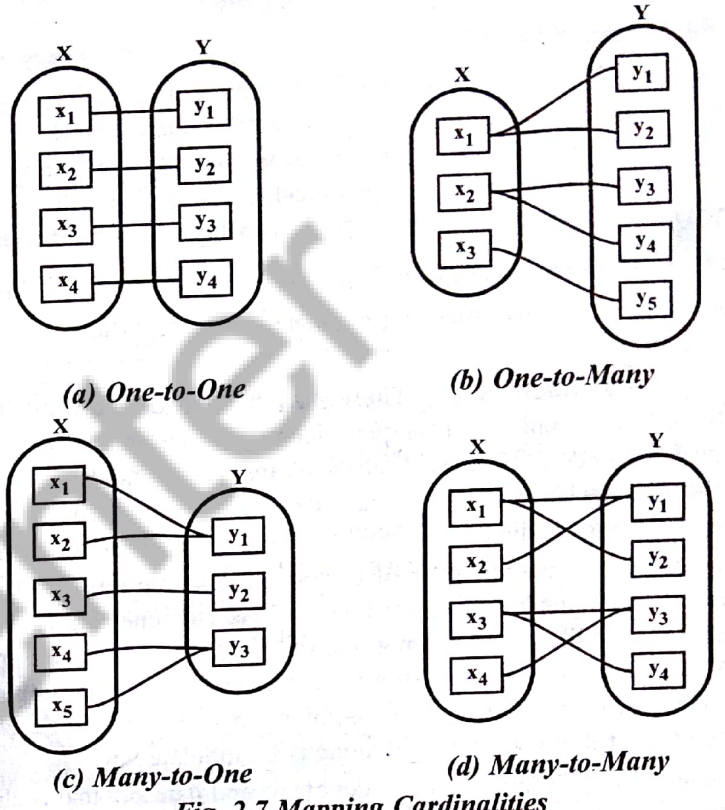
Ans. Mapping cardinalities or cardinality ratios are used to express the number of entities to which another entity can be associated through a relationship set.

For a binary relationship set R between entity sets X and Y, the mapping cardinality, must be one of the following –

(i) One-to-One – An entity in set X is associated with at most one entity in Y and an entity in Y is associated with at most one entity in X. [see fig. 2.7 (a)].

(ii) One-to-Many – An entity in X is associated with any number (zero or more) of entities in Y. However, an entity in Y can be associated with at most on entity in X. [see fig. 2.7 (b)].

(iii) Many-to-One – An entity in X is associated with at most one entity in Y. An entity in Y, however, can be associated with any number (zero or more) of entities in X. [see fig. 2.7 (c)].



(a) One-to-One *(b) One-to-Many*

(c) Many-to-One *(d) Many-to-Many*

Fig. 2.7 Mapping Cardinalities

(iv) Many-to-Many – An entity in X is associated with any number (zero or more) of entities in Y, and an entity in Y is associated with any number (zero or more) of entities in X. [see fig. 2.7 (d)].

Q.53. Discuss the naming conventions used for E-R schema diagram. ***(R.G.P.V., June 2011)***

Ans. The choice of names for entity types, roles, relationship types, and attributes is not always straightforward. One should choose names that convey, as much as possible, the meanings attached to the different constructs in the schema. We choose to use singular names for entity types, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type.

In general practice, given a narrative description of the database requirements the nouns appearing in the narrative tend to give rise to entity type names, and the verbs tend to indicate names of relationship types. Attribute names generally, arise from additional nouns which describe the nouns corresponding to entity types.

Another naming consideration involves selecting relationship names to make the E-R diagram of the schema readable from top to bottom and from left to right.

Neinvenier

Q.54. What is a key ?

Ans. A key is a single attribute or combination of two or more attributes of an entity set that is used to identify one or more instances of the set.

Therefore, a key give us the permission to identify a set of attributes that suffice to distinguish entities from each other. Keys also help to uniquely identify relationships and so distinguish relationships from each other.

Q.55. Explain various types of keys with example. (R.G.P.V., June 2015)

Ans. There are many types of keys, some of the important keys are given as follows –

(i) Candidate Key – There may be two or more attributes or combinations of attributes that uniquely identify an instance of an entity set. These attributes or combinations of attributes are called candidate keys. In such a case, we must decide which of the candidate keys will be used as the primary key. The remaining candidate keys will be considered as alternate keys.

For example, a combination of customer_name and customer_street is sufficient to distinguish among members of the customer entity set. Then, both {customer_id} and {customer_name, customer_street} are candidate keys. Although the attributes customer_id and customer_name together can distinguish customer entities, their combination does not form a candidate key, since the attribute customer_id alone is a candidate key.

The term primary key is used to denote a candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. A key (primary, candidate and super) is a property of the entity set, rather than of the individual entities. Any two individual entities in the set are prohibited from having the same value on the key attributes at the same time.

Candidate keys must be designed carefully. For example, *name* of student is obviously not sufficient, because there may be many students having same name. But the *roll_number* can be definitely identify students uniquely. So, it can be considered as candidate key. If we have more than one candidate key, then which one is to be selected and at what time ? The designer assumes that a single chosen candidate key is a primary key. We can say that, primary key is a subset of set of candidate keys and candidate key is a subset of superkeys.

(ii) Super Key – A ***superkey*** is a set of one or more attributes, taken collectively, allow us to identify uniquely an entity in the entity set. It means that if we add additional attributes to a primary key, the resulting combination will still uniquely identify an instance of the entity set. Thus, a primary key is a minimum superkey. For example, the customer_id attribute of the entity set customer is sufficient to distinguish one customer entity from another. That is why, customer_id is a superkey. Similarly, the combination of customer_name and customer_id is a superkey for the entity set customer. But the customer_name attribute of customer is not a superkey, because several people may have the same name.

(iii) Primary Key and Alternate Key – Two instances of an entity set can have the same values for all its attributes. To distinguish such instances an attribute is introduced, which is unique and identify each instance of the entity set is called, a ***primary key.*** A relation can have more than one candidate keys. In this case, at least one of them should be chosen as the primary key and others are then called alternate keys.

For example, if we consider a relation ELEMENT with attributes NAME, SYMBOL and ATOMIC.

Then we can choose SYMBOL as the primary key. NAME and ATOMIC# which are definitely candidate keys are considered as alternate keys.

In the cases where relation have only one candidate key, that only key can be considered as primary key. In other words, we can consider it as an election, in which many candidates are there requesting for vote, so as to get selected and become the president.

Each and every candidate fulfil the conditions to fill the form of election to become president. But only one of them is selected as president.

So now simulating it with our example, one candidate is selected and assigned as primary key, while all other candidates are alternate keys.

(iv) Foreign Keys – A foreign key is a set of attributes of one relation R_2, whose values are required to match values of some candidate key of some other relations R_1.

For example, see the fig. 2.8 which shows two relations *customer* and *order* where we store customer details and order details in two separate relations for reducing redundancy.

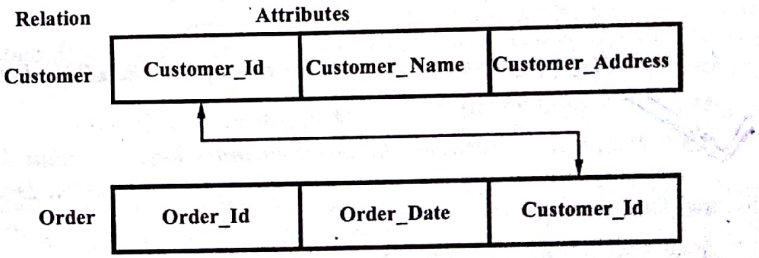


Fig. 2.8 Example of Foreign Key

The attribute *'customer_id'* in relation *'order'* is the foreign key. It allows the association of order with the respective entity in relation *'customer'*.

This example serves to motivate this definition.

Let R_2 be a relation. Then the foreign key in R_2 is a set of attributes of R_2 say F_K such that –

(a) There exists a relation R_1 (R_1 and R_2 not necessarily distinct) with a candidate key CK, and

(b) For all time, each value of F_K in the current value of R_2 is identical to the value of CK in some tuple in the current value of R_1.

Points related to foreign key are as follows –

(a) As foreign key is *simple* or *composite* according as the candidate key it matches it is simple or composite.

(b) Each attribute of a given foreign key must have the same name and type as the corresponding component of the matching candidate key.

(c) As given in above example, if order have order for that Customer_ID, that does exist in the relation Customer. To have a check over this condition, we have referential integrity constraint. A referential integrity constraint is a rule, that maintains consistency among the records of two relations. The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key in the order relation or else the value of foreign key attribute is NULL.

(v) Composite Key – There are situation when a single column cannot constitute a key. This is because a single column uniquely identify every row in a table. Instead, we need to have two or more columns together in order to identify every row in the table uniquely. A key consisting of two or more columns is called as a composite key.

For example, consider the relation Supplier (sup_id, part_id, Qty) tells us which supplier sells which part. As we know neither the sup_id nor the part_id can identify a row in the relation uniquely because a single supplier supply more than one part and also, a single part is supplied by more than one supplier. However, the two of them i.e., sup_id and part_id, together can easily identify any row in the table uniquely. Hence, it is a composite key.

Q.56. Explain super key wih suitable example. (R.G.P.V., June 2016)

Ans. Refer to Q.55 (ii).

Q.57. Distinctions between the terms primary key, candidate key.

(R.G.P.V., Dec. 2015)

Ans. Refer to Q.55.

Q.58. Discuss the candidate key, primary key, super key, composite key and alternate key.

(R.G.P.V., Dec. 2011, 2013)

Or

Explain primary key, super keys, candidate keys with examples.

(R.G.P.V., June 2013)

Ans. Refer to Q.55.

Q.59. What is the difference between a key and a super key ? Why do we designate one of the candidate keys of a relation to be the primary key ?

(R.G.P.V., June 2011)

Ans. Refer to Q.54 and Q.55.

Q.60. What are the major aspects in which the relational model deals ? Explain the following keys with example –

(i) Primary key (ii) Candidate key

(iii) Super key (iv) Foreign key.

(R.G.P.V., June 2009)

Ans. The major aspects in which the relational model deals are –

(i) Structural Aspect – The data in the database is perceived by the user as tables and nothing but tables.

(ii) Integrity Aspect – Those tables satisfy certain integrity constraints.

(iii) Manipulative Aspect – The operators available to the user for manipulating those tables, e.g., for purpose of data retrieval, are operators that derive tables from tables. Of those operators, three particularly important ones are ***restrict, project*** and ***join.***

Another aspect of the relational model, namely the formal constraint of functional and multivalued dependencies, these dependencies are used to develop a relational database design theory based on the concept known as ***normalization.***

Primary Keys, Candidate Keys, Super Keys and Foreign Keys – Refer to Q.55.

Q.61. Define primary key, candidate key, foreign key and super key with an example. Discuss the types of integrity constraints that must be checked for the update operations Insert and Delete. Give examples.

(R.G.P.V., June 2008)

Or

Discuss different update operations that can be performed on relations.

(R.G.P.V., June 2014)

Ans. **Primary Key, Candidate Key, Foreign Key and Super Key** – Refer to Q.55.

Update Operations and Dealing with Constraint Violations – There are three basic update operations on relations –

(i) Insert (ii) Delete (iii) Modify.

Insert is used to insert a new tuple or tuples in a relation. ***Delete*** is used to delete tuples; and ***Update*** (or modify) is used to change the values of some attributes in existing tuples. Whenever update operations are applied, the integrity constraints specified on the relational database schema should not be violated.

(i) Insert Operation – The insert operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R. Insert can violate any of the four types of constraints. Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain. Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R). Entity integrity can be violated if the primary key of the new tuple t is null. Referential integrity can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.

Some examples to illustrate this discussion are as follows –

(a) Insert <'Cecilia', 'F', 'Kolonsky', null, '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, null, 4 > into EMPLOYEE.

This insertion violates the entity integrity constraint (null for the primary key SSN), so it is rejected.

(b) Insert <'Alicia', 'J', 'Zelaya', '999887777', '1960-04-05', '6357 Windy lane, Katy, TX', F, 28000, '987654321', 4 > into EMPLOYEE.

This insertion violates the key constraint because another tuple with the same SSN value already exists in the EMPLOYEE relation, and so it is rejected.

(c) Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 windswept, katy, TX', F, 28000, '987654321', 7 > into EMPLOYEE.

This insertion violates the referential integrity constraint specified on DNO because no DEPARTMENT tuple exists with DNUMBER = 7.

(d) Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, null, 4 > into EMPLOYEE.

This insertion is accepted, because it satisfies all constraints.

If an insertion violates one or more constraints, the default option is to reject the insertion. In this case, it would be useful if the DBMS could explain to the user why the insertion was rejected. Another option is to attempt to correct the reason for rejecting the insertion, but this is not used for violations caused by insert; rather, it is used in correcting violations for delete and update. The following examples show how this option may be used for insert violations. In operation (a), the DBMS could ask the user to provide a value of SSN and could accept the insertion if a valid SSN value were provided. In operation (c), the DBMS could either ask the user to insert a DEPARTMENT tuple with DNUMBER = 7 and could accept the insertion only after such an operation was accepted.

(ii) Delete Operation – The delete operation can violate only referential integrity, if the tuple being deleted is referenced by the foreign keys from other tuples in the database. Here are some examples –

(a) Delete the WORKS_ON tuple with ESSN = '999887777' and PNO = 10.

This deletion is acceptable.

(b) Delete the EMPLOYEE tuple with SSN = '999887777'.

This deletion is not acceptable, because tuples in WORKS_ON refer to this tuple. Thus, if the tuple is deleted, referential integrity is violated.

(c) Delete the EMPLOYEE tuple with SSN = '333445555'.

This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON relations.

Three operations are present if a deletion operation causes a violation. The first option is to *reject the deletion*. The second option is to *attempt to cascade* (or *propagate*) *the deletion* by deleting tuples that reference the tuple that is being deleted. For example, in operation (b), the DBMS could automatically delete the offending tuples from WORKS_ON with ESSN = '999887777'. A third option is to modify the referencing attribute values that cause the violation, each such value is either set to null or changed to reference another valid tuple.

(iii) Update Operation – The ***update*** operation is used to update the values of one or more attributes in a tuple (or tuples) of some relation R. It is essential to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified. Here are some examples.

(a) Update the SALARY of the EMPLOYEE tuple with SSN = '999887777' to 28000.

This is acceptable.

(b) Update the DNO of the EMPLOYEE tuple with SSN = '999887777' to 1.

This is acceptable.

(c) Update the DNO of the EMPLOYEE tuple with SSN = '999887777' to 7.

This is unacceptable, because it violates referential integrity.

Updating an attribute that is neither a primary key nor a foreign key usually causes no problems; the DBMS need only check to confirm that the new value is of the correct data type and domain. Modifying a primary key value is similar to deleting one tuple and inserting another in its place. If a foreign key attribute is modified, the DBMS must make sure that the new value refers to an existing tuple in the referenced relation (or is null).

Q.62. How foreign key is useful to maintain referential integrity ?
(R.G.P.V., June 2016)

Ans. The concept of foreign key is useful to define referential integrity more formally. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R_1 and R_2. A set of

attributes FK in relation schema R_1 is a foreign key of R_1 that references relation R_2 if it satisfies the following two rules –

(i) The attributes in FK have the same domain(s) as the primary key attributes PK of R_2; attributes FK are said to reference to refer to the relation R_2.

(ii) A value of FK in a tuple t_1 of the current state r_1 (R_1) either occurs as a value of PK for some tuple t_2 in the current state r_2 (R_2) or is null. In the former case, we have t_1 [FK] = t_2 [PK], and we say that the tuple t_1 references or refers to the tuple t_2. R_1 is called the referencing relation and R_2 is the referenced relation.

In a database of many relations, there are many referential integrity constraints. To specify these constraints, it is first necessary to have a clear understanding of the meaning or role that each set of attributes plays in the various relation schemas of the database. Referential integrity constraints arise from the relationships among the entities represented by the relation schemas. For example, consider the database shown in fig. 2.9. In the employee relation, the attribute DNO refers to the department for which an employee works. Hence, we designate DNO to be a foreign key of EMPLOYEE, referring to the DEPARTMENT relation. This means that a value of DNO in any tuple t_1 of the EMPLOYEE relation must match a value of the primary key of DEPARTMENT – the DNUMBER attribute – in some tuple t_2 of the DEPARTMENT relation, or the value of DNO can be null if the employee does not belong to a department. In fig. 2.9 the tuple for employee 'John Smith' references the tuple for the 'Research' department indicating that 'John Smith' works for this department.

FNAME	LNAME	SSN	ADDRESS	SEX	SUPERSSN	DNO
John	Smith	123456789	731 Fondren, Houston, TX	M	333445555	5
Franklin	Wong	333445555	638 Voss, Houston, TX	M	888665555	5
Alicia	Zelaya	999887777	3321 Castle, Spring, TX	F	987654321	4
Jennifer	Wallace	987654321	291 Berry, Bellaire, TX	F	888665555	4
Ramesh	Narayan	666884444	975 Fire Oak, Humble, TX	M	333445555	5
Joyce	English	453453453	5631 Rice, Houston, TX	F	333445555	5
Ahmad	Jabbar	987987987	980 Dallas, Houston, TX	M	987654321	4
James	Borg	888665555	450 Stone, Houston, TX	M	Null	1

EMPLOYEE

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPARTMENT

Fig. 2.9 One Possible Relational Database State Corresponding to the Schema COMPANY

A foreign key can also refer to its own relation. For example, the attribute SUPERSSN in EMPLOYEE refers to the supervisor of an employee; this is another employee, represented by a tuple in the EMPLOYEE relation. Hence, SUPERSSN is a foreign key that references the EMPLOYEE relation itself.

Q.63. Discuss the design issues of an E-R database schema.

Ans. The notions of an entity set and a relationship set are not precise. So, it is possible to define a set of entities and the relationships among them in a number of different ways. Thus, the basic design issues of an E-R database schema are as follows –

(i) Entity Set Versus Attributes – Let, there is a entity set employee with attributes employee_name and telephone_number. But, telephone is an entity in its own right with attributes telephone_number and location. Based on this fact, employee entity set can be redefined as –

(a) The employee entity set with attribute employee_name.

(b) The telephone entity set with telephone_number and locations.

(c) The relationship set emp_telephone, which denotes the association between employees and the telephones that they have.

In the above two definitions of an employee, treating a telephone as an attribute telephone_number implies that employees have one telephone number each. Also, treating a telephone as an entity telephone allows employees to have several telephone numbers associated with them. However, we can instead define telephone_number as a multivalued attribute to allow multiple telephones per employee.

The main difference is that treating a telephone as an entity better models a situation where extra information about a telephone is to be kept such as its location, its type, or who all share the telephone. Thus, treating telephone as an entity is more general than treating it as an attribute and is appropriate when the generality may be useful.

In contrast, it will not be appropriate to treat the attribute employee_name as an entity. It is difficult to argue that employee_name is an entity in its own right. So, it is appropriate to have employee_name as an attribute of the employee entity set.

Here, there are two questions arise – What constitute an attribute ? And what constitutes an entity set ? The distinctions depend on the structure of the real-world enterprise being modeled, and on the semantics associated with the attribute in question.

A common mistake is to use the primary key of an entity set as an attribute of another entity set, instead of using a relationship. For example, it is incorrect to model customer_id as an attribute of loan even if each loan had only one customer. The relationship borrower is the correct way to represent the

connection between loans and customers due to explicit connection through an attribute.

Another mistake is that people commit is to designate the primary key attributes of the related entity sets as attributes of the relationship set.

(ii) Entity Sets versus Relationship Sets – It is not always clear that whether an object is best expressed by an entity set or a relationship set. For example, a bank loan can be modeled as an entity. Another alternative is to model a loan as a relationship between customers and branches instead of an entity, with loan_number and amount as attributes.

If every loan is held by exactly one customer and is associated with exactly one branch, then satisfactory design can be found where a loan is represented as a relationship. However, a situation in which several customers holding a loan jointly, cannot be represented conveniently with this design. To handle such a situation, we must define a separate relationship for each holder of the joint loan. Then, we must replicate the values for the descriptive attributes loan_number and amount in each such relationship. Each such relationship must have the same value for the descriptive attributes loan_number and amount.

Here, there are two problems arise as a result of replication –

(a) The data are stored multiple times, wasting storage space.

(b) Updates potentially leave the data in an inconsistent state, where the values differ in two relationships for attributes that are supposed to have the same value.

This replication is avoided by using normalization.

One guideline in determining whether to use an entity set or a relationship is to designate a relationship set to describe an action that occurs between entities. This approach is also useful in deciding whether certain attributes may be more appropriately expressed as relationships.

(iii) Binary versus n-ary Relationship Sets – Relationships in databases are often binary. The relationships that are nonbinary, can be represented by several binary relationships. For example, one can create a ternary relationship parent, relating a child to his/her mother and father. This relationship can also be represented by two binary relationships, mother and father, relating a child to his/her mother and father separately.

In fact, it is always possible to replace a nonbinary (n-ary, for $n > 2$) relationship set by a number of distinct binary relationship sets. For simplicity, consider the abstract ternary ($n = 3$) relationship set R, relating entity sets A, B, and C. The relationship R is replaced by an entity set E, and by creating three relationship sets

R_A, relating E and A
R_B, relating E and B
R_C, relating E and C

If the relationship set R had any attributes, these are assigned to entity set E. Further, a special identifying attribute is created for E to distinguish different entities in an entity set on the basis of their attribute values. For each relationship (a_i, b_i, c_i) in the relationship set R, we create a new entity e_i in the entity set E. Then, in each of the three new relationship sets, we insert a relationship as follows –

(a) (e_i, a_i) in R_A (b) (e_i, b_i) in R_B (c) (e_i, c_i) in R_C.

Q.64. Explain the representation of E-R diagram using a suitable example.

Or

Discuss the conventions for displaying an E-R schema as an E-R diagram. ***(R.G.P.V., June 2010)***

Or

Describe the components of entity relationship diagram with suitable example. ***(R.G.P.V., Dec. 2014)***

Ans. The overall logical structure of a database can be expressed graphically by an E-R diagram. Because of its pictorial representation, it is easy to understand. Such a diagram consists of following main components –

(i) Rectangles represent entity sets
(ii) Ellipses represent attributes
(iii) Diamonds represent relationship sets
(iv) Lines link attributes to entity sets and entity sets to relationship sets
(v) Double ellipses represent multivalued attributes
(vi) Dashed ellipses denote derived attributes
(vii) Double lines indicate total participation of an entity in the relationship set
(viii) Underlines denote the primary key attribute in an entity set
(ix) Double rectangles represent weak entity sets.

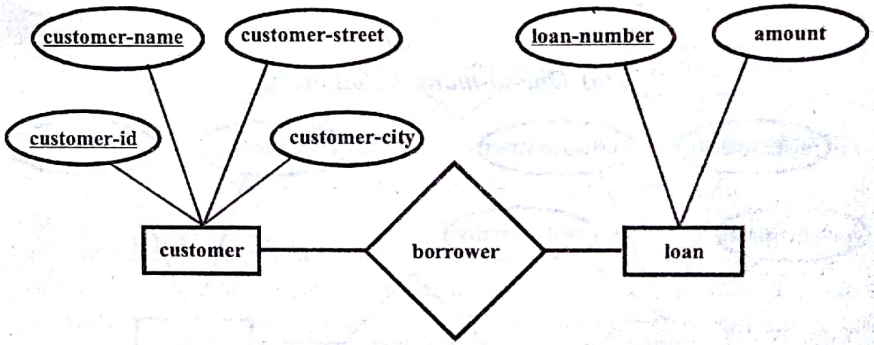


Fig. 2.10 E-R Diagram Corresponding to Customer and Loan

Fig. 2.10 shows the entity-relationship (E-R) diagram consists of two entity sets, customer and loan, related through a binary relationship set borrower.

The attributes associated with customer are customer-id, customer-name, customer-street and customer-city. The attributes associated with loan are loan-number and amount. In fig. 2.10, attributes of an entity set that are members of the primary key are underlined.

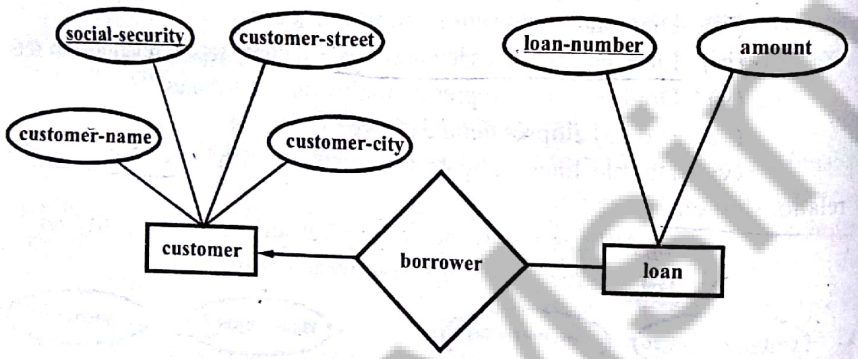
The relationship set borrower may be many-to-many, one-to-many, many-to-one and one-to-one. To distinguish between these types, either a directed line (→) or an undirected line (–) is drawn between the relationship set and the entity set.

Specification about directed and undirected lines are as follows –

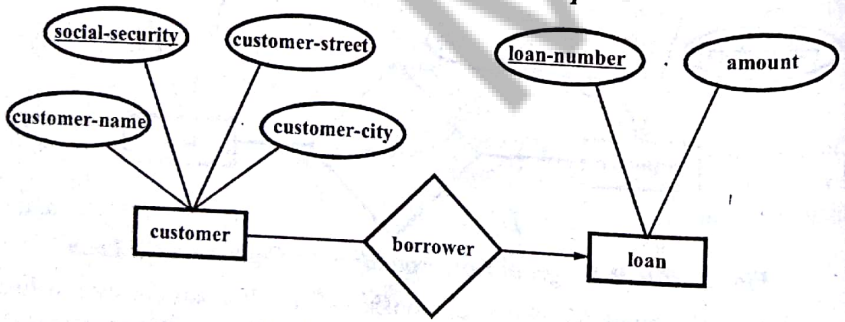
(i) A directed line from relationship set ***borrower*** to the entity set ***loan*** specifies that ***borrower*** is either one-to-one or many-to-one relationship set, from ***customer*** to ***loan***; ***borrower*** cannot be many-to-many or one-to-many relationship set from ***customer*** to ***loan***.

(ii) Similarly, an undirected line from the relationship set ***borrower*** to the entity set ***loan*** specifies, that ***borrower*** is either a many-to-many or one-to-many relationship set, from ***customer*** to ***loan***.

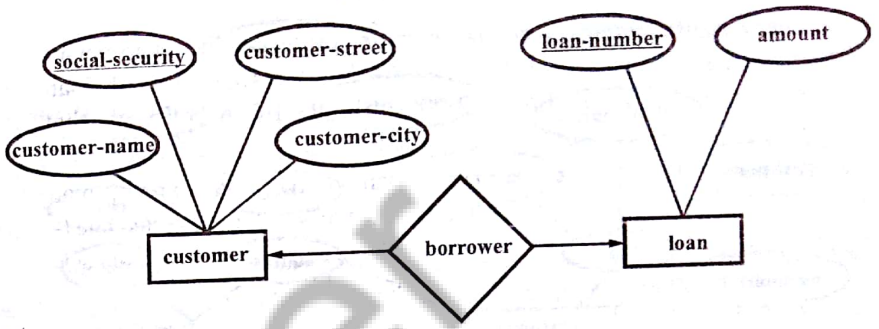
In fig. 2.10, ***borrower*** relationship set is many-to-many relationship. If the relationship set ***borrower*** were one-to-many from customer to loan, then the line from ***borrower*** to ***customer*** would be directed, with an arrow pointing to ***customer*** entity set as shown in fig. 2.11 (a).



(a) One-to-many Relationship



(b) Many-to-one Relationship



(c) One-to-one Relationship

Fig. 2.11 Relationships

Similarly, if the ***borrower*** relationship set were many-to-one from customer to loan, then line from ***borrower*** to ***loan*** would be directed i.e., arrow pointing to ***loan*** entity set, as shown in fig. 2.11 (b).

Finally, if the ***borrower*** relationship set were to one-to-one, then both lines from ***borrower*** to ***customer*** and from ***borrower*** to ***loan*** would be directed i.e., arrow pointing to ***customer*** and ***loan*** entity sets, as shown in fig. 2.11 (c).

If a relationship set has also some attributes associated with it, then we link these attributes to that relationship set. For example, in fig. 2.12, the descriptive attribute ***access-date*** has been attached to the relationship set depositor to specify the most recent data on which a customer accessed that account.

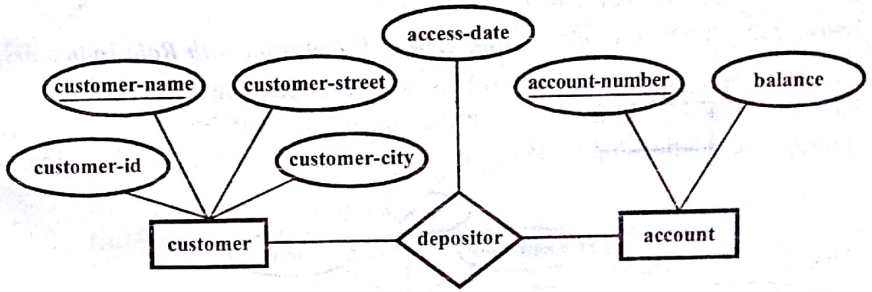


Fig. 2.12 E-R Diagram with an Attribute attached to a Relationship Set

Fig. 2.13 shows how composite attributes are represented in the E-R notation. Here, composite attribute name, with component attributes first-name, middle-name and last-name replaces the attribute customer-name of customer. Also, a composite attribute address, whose component attributes are street, city, state and zip-code replaces the attributes customer-street and customer-city of customer. The attribute street is itself a composite attribute whose component attributes are street-number, street-name and apartment-number.

MSInvester

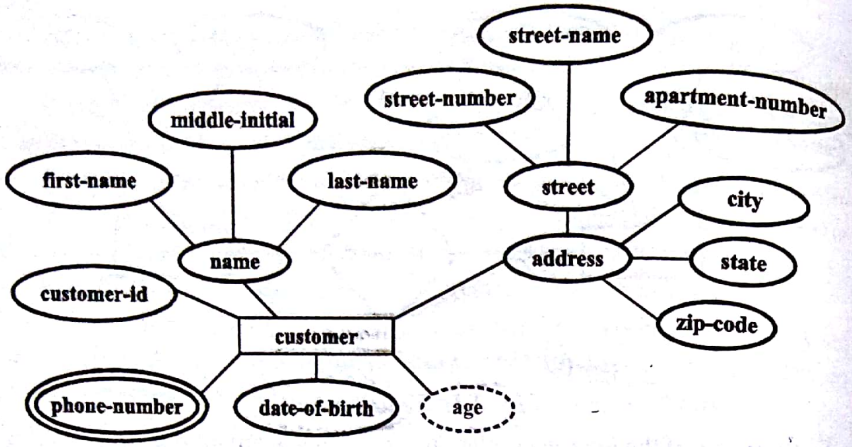


Fig. 2.13 E-R Diagram with Composite, Multivalued and Derived Attributes

Fig. 2.13 also shows a multivalued attribute phone-number, depicted by double ellipses and desired attribute age, depicted by a dashed ellipse.

Roles of an E-R diagram can also be indicated by lines that connect diamonds to rectangles. Fig. 2.14 shows the role indicators, manager and worker, between the ***employee*** entity set and ***works-for*** relationship set.

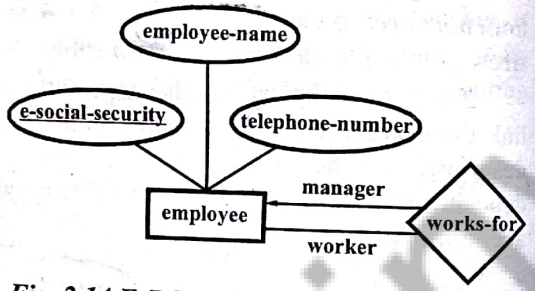


Fig. 2.14 E-R Diagram with Role Indicators

Normally, we have binary relationship between entity sets, but we also have examples in which three entities customer, loan and branch, related through the relationship CLB i.e., a ternary relationship set (see fig. 2.15).

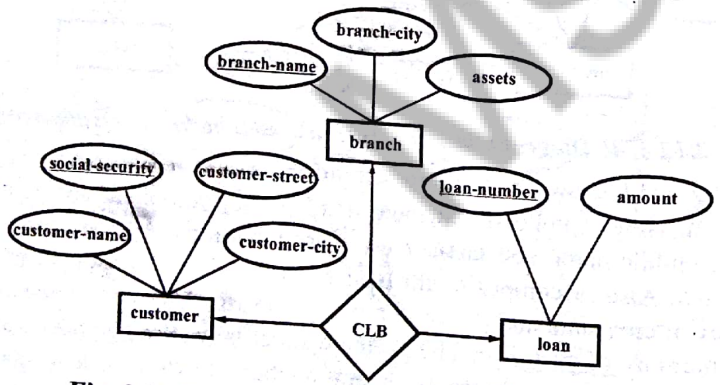


Fig. 2.15 E-R Diagram with Ternary Relationship

This diagram indicates that customer may have several loans and that loan may belong to several different customers. The arrow pointing to branch indicates that, each customer-loan pair is associated with a specific branch of bank.

Q.65. Draw an E-R diagram corresponding to customers and loans. ***(R.G.P.V., June 2015)***

Ans. Refer to Q.64.

Q.66. List the possible types of relations that may exist between two entities. How would you realize that into tables for a binary relation ? ***(R.G.P.V., Dec. 2012)***

Ans. Refer to Q.64.

Q.67. Differentiate between strong entity and weak entity. ***(R.G.P.V., Nov./Dec. 2007)***

Or

Define weak entity and strong entity. ***(R.G.P.V., Dec. 2013)***

Or

What do you mean by weak entity set ? ***(R.G.P.V., Dec. 2014)***

Ans. An entity set that does not have sufficient attributes to form a primary key is called a ***weak entity set***. In contrast, an entity set that has a primary key is called a ***strong entity set***.

For a weak entity set has to be meaningful, it must be associated with another entity set, called the ***identifying*** or ***owner entity set***. Every weak entity can be associated with an identifying entity i.e., the weak entity set is said to ***own*** the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship. The identifying relationship is many to one from the weak entity set to the identifying entity set and the participation of the weak entity set in the relationship is total.

Q.68. Justify the following statements –

(i) Handling null values is difficult

(ii) Relations must have a key

(iii) Weak entities do not have their own key attributes. ***(R.G.P.V., June 2010)***

Ans. (i) When an entity does not have a value for attribute then this attribute stores a null value which indicate 'not applicable' – that is, that the value does not exist for the entity. For example, the absence of middle name. Null can also specifies that an attribute value is unknown. An unknown value may be either missing or not known.

To deal with null values, there is more than one possible way and as a result our definitions can sometimes be arbitrary. Therefore operations and comparisons on null values should be avoided, where possible.

Since the special value null indicates "value unknown or nonexistent" any arithmetic operations involving null values must return a null result. Similarly,

any comparisons involving a null value evaluate to special value unknown. We cannot say for sure whether the result of the comparison is true or false, so we say that the result is the new truth value unknown.

(ii) We must have a way to specify how entities within a given entity set are distinguished. From a database perspective, however, the difference among them must be expressed in terms of their attributes.

Therefore, the values of the attribute values of an entity must be such that they can uniquely identify the entity. In other words, we can say that no two entities in an entity set are permitted to have exactly the same value for all attributes.

A key allows us to identify a set of attributes that suffice to distinguish entities from each other. Keys also help uniquely identify relationships, and thus distinguish relationships from each other.

(iii) Refer to Q.67.

Q.69. Give examples of the following –

(i) A many-to-many relationship in which one of the participants is another relationship.

(ii) A subtype that has an associated weak entity that does not apply to the supertype. ***(R.G.P.V., Dec. 2008)***

Ans. (i) Fig. 2.16 shows an example of an E-R diagram.

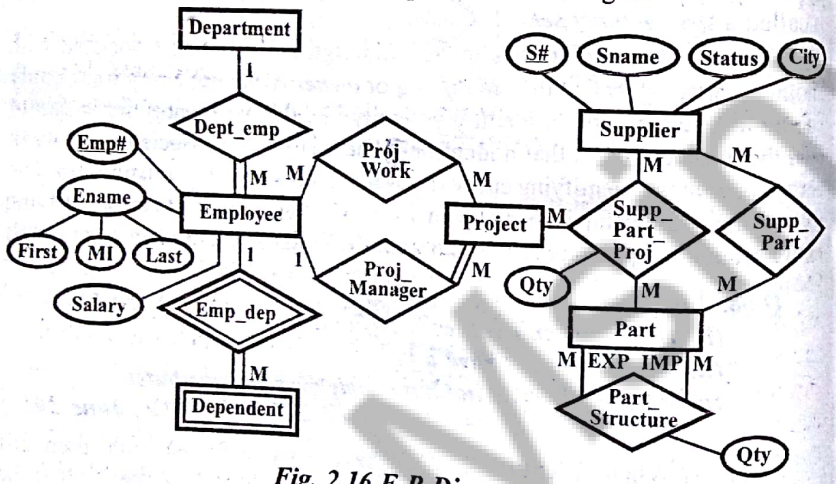


Fig. 2.16 E-R Diagram

The many-to-many (or many-to-many-to-many, etc.) relationships in the example are as follows –

Proj_Work (involving employees and projects)
Supp_Part (involving suppliers and parts)
Supp_Part_Proj (involving suppliers, parts, and projects)
Part_Structure (involving parts and parts)

Each such relationship also maps into a base relation.

(ii) Fig. 2.17 shows an example of an entity type hierarchy.

Let us concentrate on entity types Employee and Programmer. Assume that programmers have just one programming language skill. Then, the subtype Programmer maps into another base relation, PGMR say, with primary key the same as the supertype relation and with other attributes corresponding to the properties that apply only to employees who are programmers.

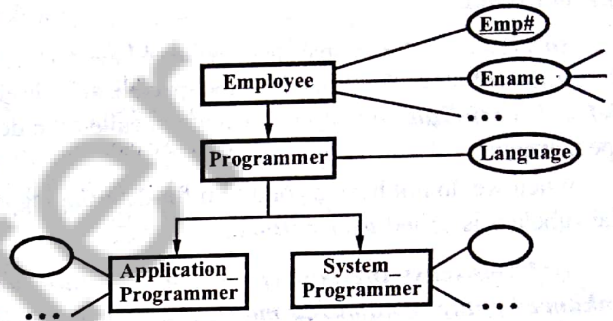


Fig. 2.17 Example of an Entity Type Hierarchy

Q.70. Explain entity subtypes and supertypes.

Or

What is subclass ? When is subclass needed in data modelling ? Explain is-a relationship. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. An entity type is used to represent both a type of entity, and the entity set or collection of entities of that exist in the database. For example, the entity type EMPLOYEE describes the type of each employee entity and also refers to the current set of EMPLOYEE entities in the COMPANY database. In many cases an entity type has numerous subgroupings of its entities that are meaningful and need to be represented explicitly because of their significance to the database application. For example, the entities that are members of the EMPLOYEE entity type may be grouped further into SECRETARY, ENGINEER, MANAGER, TECHNICIAN, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE, and so on. The set of entities in each of the latter groupings is a subset of entities that belong to the EMPLOYEE. Entity set, meaning that every entity that is a member of one of these subgroupings is also an employee. Each of these subgroupings is called a ***subclass*** of the EMPLOYEE entity type and the EMPLOYEE entity type is called the ***superclass*** for each of these subclasses.

The relationship between a superclass and any one of its subclasses is called a superclass/subclass or class/subclass relationship. For example, EMPLOYEE/SECRETARY and EMPLOYEE/TECHNICIAN are two class/subclass relationships. A class/subclass relationship is often called an IS-A (or IS-AN) relationship. We say "a SECRETARY IS-AN EMPLOYEE," "a TECHNICIAN IS-AN EMPLOYEE" and so forth. A member entity of the subclass represents the same real-world entity as some member of the superclass; for example, a SECRETARY entity 'Joan Logano' is also the

EMPLOYEE 'Joan Logano'. Hence, the subclass member is the same as the entity in the superclass, but in a distinct specific role.

Q.71. Differentiate between user defined and attribute defined specialization. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. If all subclasses in a specialization have the membership condition on the same attribute of the superclass, the specialization itself is called an ***attribute-defined specialization*** and the attribute is called the defining attribute of the specialization.

When we do not have a condition for determining membership in a class, the subclass is called ***user-defined.***

Q.72. Discuss the concept of specialization and generalization in enhanced entity-relationship model using an example.

Or

Differentiate between specialization and generalization. ***(R.G.P.V., Nov./Dec. 2007)***

Or

Explain the distinction between specialization and generalization concepts. ***(R.G.P.V., June 2012)***

Or

Compare specialization and generalization. ***(R.G.P.V., June 2014)***

Ans. Specialization is the process of defining a set of subclasses of an entity type. This entity type is called the superclass of the specialization. The set of subclasses that form a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass. For example, the set of subclasses {secretary, engineer, technician} is a specialization of the superclass employee that distinguishes among employee entities based on the job type of each entity. There may have several specializations of the same entity type based on different distinguishing characteristics. For example, another specialization of the employee entity type may be the set of subclasses {Salaried_employee, Hourly_employee}. This specialization distinguishes among employees based on the method of pay.

Fig. 2.18 shows how we represent a specialization diagrammatically in an enhanced E-R diagram. The subclasses that define a specialization are attached by line to a circle, which is connected to the superclass. The ***subset symbol*** on each line connecting a subclass to the circle indicates the direction of the superclass/subclass relationship. Attributes that apply only to entities of a particular subclass – such as typing speed of secretary – are attached to the rectangle representing that subclass. These are called ***specific*** or ***local attributes*** of the subclass. Similarly, a subclass can participate in specific relationship

types, such as the Hourly_employee class participating in the Belongs_to relationship in fig. 2.18.

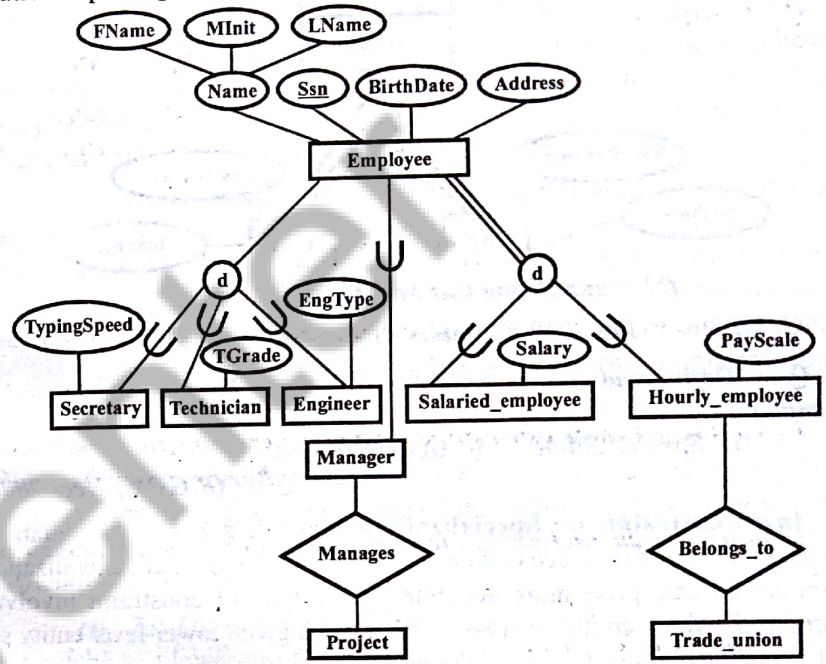
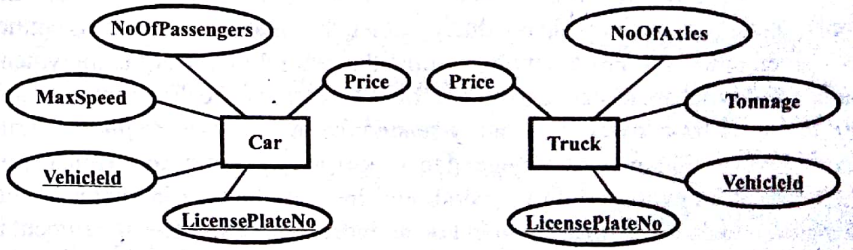


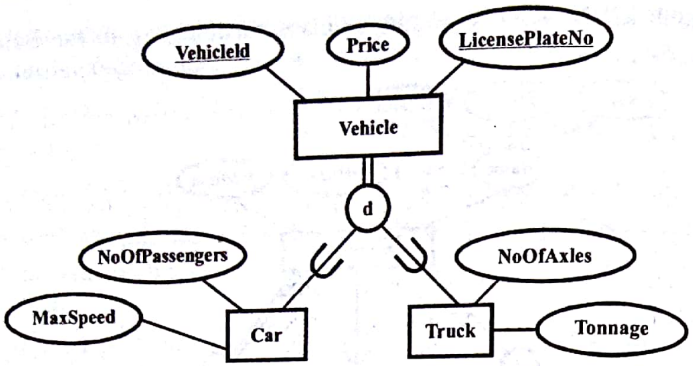
Fig. 2.18 Enhanced E-R Diagram Notation for Representing Specialization and Subclasses

Generalization is the reverse process of specialization in which we suppress the differences among several entity types, identify their common features, and generalize them into a single superclass of which the original entity types are special subclasses. For example, consider the entity types car and truck shown in fig. 2.19 (a). They can be generalized into the entity type vehicle as shown in fig. 2.19 (b). Now, both car and truck are subclasses of the generalized superclass vehicle.

In fig. 2.19, we can view {car, truck} as specialization of vehicle, rather than viewing vehicle as a generalization of car and truck. Similarly, in fig. 2.18 we can view employee as a generalization of secretary, technician and engineer.



(a) Two Entity Types Car and Truck



(b) Generalizing Car and Truck into Vehicle

Fig. 2.19 Examples of Generalization

Q.73. Define and discuss main constraints on the following by giving examples –

(i) Specialization (ii) Generalization.

(R.G.P.V., Dec. 2006)

Ans. **Constraints on Specialization/Generalization** – The database designer can choose to place certain constraints on a particular generalization to model an enterprise more accurately. One type of constraint involves determining which entities can be members of a given lower-level entity set such membership may be one of the following –

(i) Condition-defined – In condition-defined lower-level entity sets, membership is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate. For example, suppose that the higher-level entity set ***account*** has the attribute ***account-type.*** All ***account*** entities are evaluated on the defining *account-type* attribute. Only those entities that satisfy the condition account-type = "savings account" are allowed to belong to the lower-level entity set person. All entities that satisfy the condition ***account-type = "checking account"*** are included in ***checking account.*** Since all the lower-level entities are evaluated on the basis of the same attribute, this type of generalization is said to be ***attribute-defined.***

(ii) User-defined – User-defined lower-level entity sets are not constrained by a membership condition; rather, the database user assigns entities to a given entity set. For example, assume that after 3 months of employment, bank employees are assigned to one of four work teams. We therefore represent the teams as four lower-level entity-sets of the higher-level ***employee*** entity set. A given employee is not assigned to a specific team entity automatically on the basis of an explicit defining condition. Instead, the user in charge of this decision makes the team assignment on an individual basis. The assignment is implemented by an operation that adds an entity to an entity set.

A second type of constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization. The lower-level entity sets may be one of the following –

(i) Disjoint – A disjointness constraint requires that an entity belong to no more than one lower-level entity set. For example, an *account* entity can satisfy only one condition for the *account-type* attribute; an entity can be either a savings account or a checking account, but cannot be both.

(ii) Overlapping – In overlapping generalizations, the same entity may belong to more than one lower-level entity set within a single generalization. For an illustration, consider the employee work team example, and assume that certain managers participate in more than one work team. A given employee may therefore appear in more than one of the team entity sets that are lower-level entity sets of employee. Therefore, the generalization is overlapping.

A final constraint, the **completeness constraint** on a generalization or specialization, specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization. This constraint may be one of the following –

(i) Total Generalization or Specialization – Each higher-level entity must belong to a lower-level entity set.

(ii) Partial Generalization or Specialization – Some higher-level entities may not belong to any lower-level entity set.

The ***account*** generalization is total – All account entities must be either savings account or a checking account. Because the higher-level entity set arrived at through generalization is generally composed of only those entities in the lower-level entity sets, the completeness constraint for a generalized higher-level entity set is usually total. When the generalization is partial, a higher-level entity is not constrained to appear in a lower-level entity set. The work team entity sets illustrate a partial specialization. Since employees are assigned to a team only after 3 months on the job, some ***employee*** entities may not be members of any of the lower-level team entity sets.

Q.74. Explain the following constraints while designing E-R diagram.

(i) Total and partial constraints

(ii) Disjoint and partial constraints

(iii) Condition defined and user defined constraints.

(R.G.P.V., June 2013)

Ans. Refer to Q.73.

Q.75. What is meant by the aggregation ? Explain with example.
(R.G.P.V., Dec. 2010)

Ans. One limitation of E-R model is that it cannot express relationships among relationships. For example, let us the ternary relationship works-on between a employee, branch and job. Now, suppose that we want to record managers for tasks performed by an employee at a branch. That is, we want to record managers for (employee, branch, job) combinations. Let us assume that there is an entity set manager.

One solution for representing this relationship is to create a quaternary relationship ***manages*** between ***employee***, ***branch***, ***job*** and ***manager***. Thus, the resulting E-R diagram will be as shown in fig. 2.20.

However, there is redundant information in fig. 2.20, since every employee, branch, job combination in manages is also in works_on.

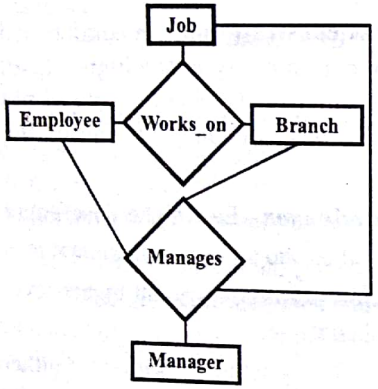


Fig. 2.20 E-R Diagram with Redundant Relationships

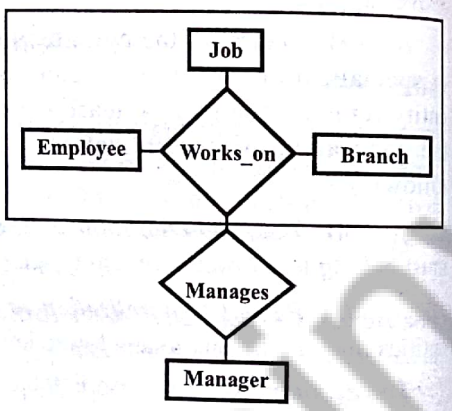


Fig. 2.21 E-R Diagram with Aggregation

The best way to model such a situation is aggregation. So, aggregation is an abstraction through which relationships are treated as high-level entities. Thus, we regard the relationship set works_on as a higher-level entity set called works_on. Such an entity set is treated in the same manner as is any other entity set. Therefore, a binary relationship manages is created between works_on and manager to represent who manages what tasks as shown in fig. 2.21.

Q.76. Explain the following –
(i) Aggregation (ii) Strong and weak entity set
(iii) Generalization/specialization.

(R.G.P.V., Dec. 2017)

Ans. (i) Aggregation – Refer to Q.75.
(ii) Strong and Weak Entity Set – Refer to Q.67.
(iii) Generalization/Specialization – Refer to Q.72.

Q.77. Define the following with respect to an E-R diagram. Explain the manner in which each is mapped to a table. Illustrate with an example –
(i) Relationship set (ii) Aggregation (iii) Multivalued attribute.
(R.G.P.V., June 2008)

Ans. **Relationship Set** – Refer to Q.51 and Q.64.

Aggregation – Refer to Q.75.

Multivalued Attribute – Refer to Q.53 (Unit-I) and Q.64.

Tabular Representation of Relationship Sets – Let R be a relationship set, let $a_1, a_2,..., a_m$ be the set of attributes formed by the union of the primary keys of each of the entity sets participating in R, and let the descriptive attributes (if any) of R be $b_1, b_2,...., b_n$. This relationship set is represented by a table called R with one column for each attribute of the set

$$\{a_1, a_2,....,a_m\} \cup \{b_1, b_2,....,b_n\}$$

For example, consider the relationship set *borrower* in the E-R diagram of fig. 2.10. This relationship set involves the following two entity sets –

(i) *customer,* with the primary key *customer_id*

(ii) *loan,* with the primary key *loan_number.*

Since the relationship set has no attributes, the *borrower* table has two columns, labeled *customer_id* and *loan_number,* as shown in fig. 2.22.

Tabular Representation of Aggregation – Transforming an E-R diagram containing aggregation to a tabular form is straightforward. Consider the diagram of fig. 2.21. The table for the relationship set *manages* between the aggregation of *works_on* and the entity set *manager* includes a column for each attribute in the primary keys of the entity set *manager* and the relationship set works_on. It would also include a column for any descriptive attributes, if they exist, of the relationship set *manages*. Then we have transform the relationship sets and entity sets within the aggregated entity.

customer_id	loan_number
019-28-3746	L-11
019-28-3746	L-23
244-66-8800	L-93
321-12-3123	L-17
335-57-7991	L-16
555-55-5555	L-14
677-89-9011	L-15
963-96-3963	L-17

Fig. 2.22 The borrower Table

Tabular Representation of Multivalued Attributes – Generally, attributes in an E-R diagram map directly into columns for the appropriate tables. However, multivalued attributes are an exception; new tables are created for these attributes.

For a multivalued attribute M, we create a table T with a column C that corresponds to M and columns corresponding to the primary key of the entity set or relationship set of which M is an attribute. For example, consider the E-R diagram in fig. 2.23. The diagram includes the multivalued attribute *dependent_name*. For this multivalued attribute, we create a table *dependent_name*, with columns *dname*, referring to the *dependent_name* attribute of *employee* and *employee_id*, representing the primary key of the entity set *employee*. Each dependent of an employee is represented as a unique row in the table.

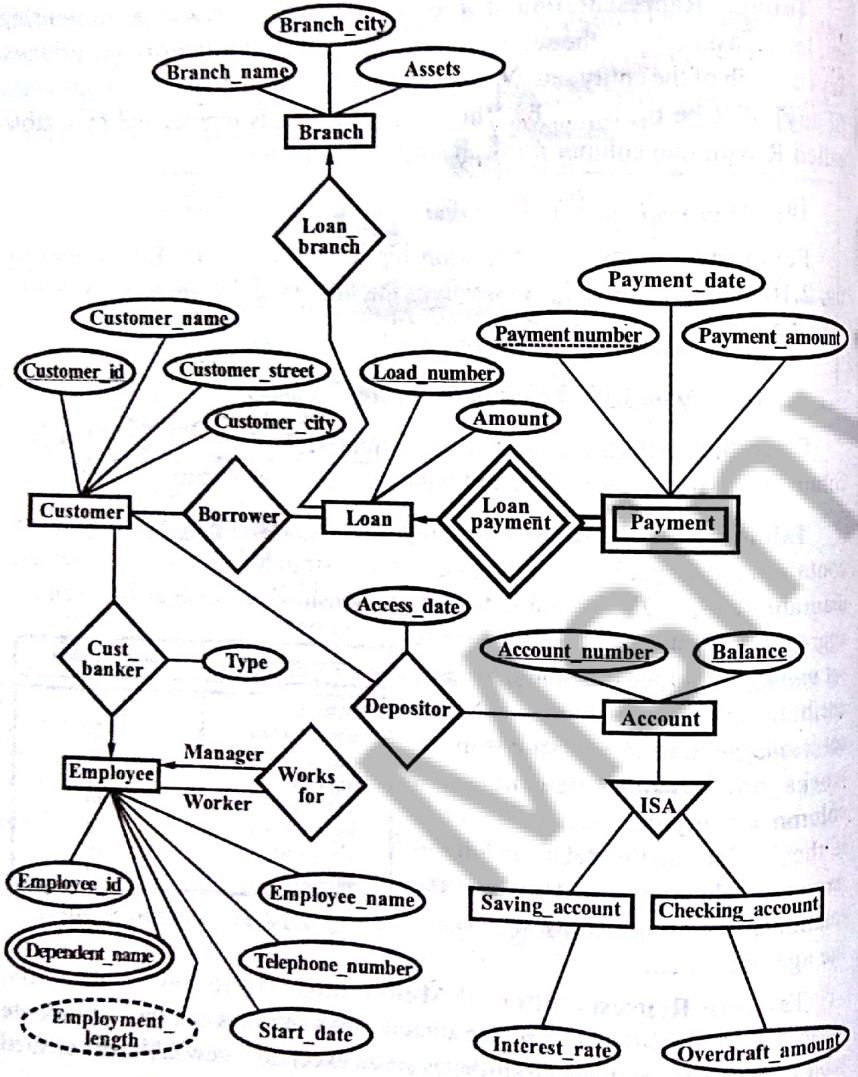


Fig. 2.23 E-R Diagram for a Banking Enterprise

NUMERICAL PROBLEMS

Prob.1. Construct an E-R diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. ***(R.G.P.V., Dec. 2009)***

Or

Construct an ER diagram for a car insurance company that has a set of customers, each of whom owns one/more cars. Each car has associated with it zero to any number of recorded accidents.

(R.G.P.V., Dec. 2012)

Sol. The E-R diagram is shown in fig. 2.24.

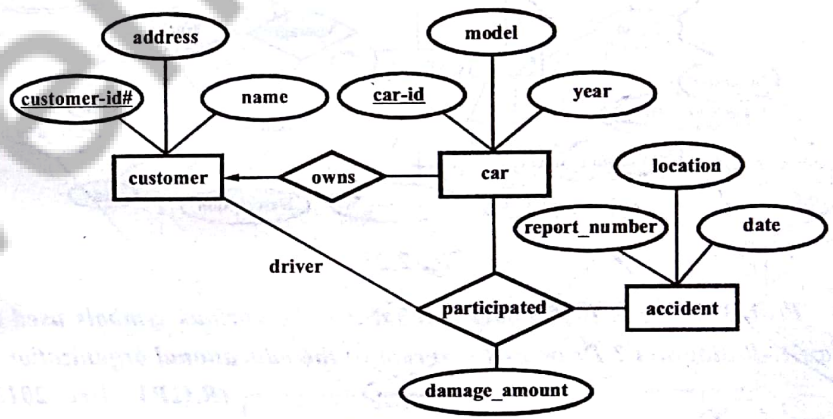


Fig. 2.24 E-R Diagram

The tables for this diagram can be written as relations as follows –

customer (customer_id #, name, address)

car (license, model, year)

accident (report_number, date, location)

owns (driver_id #, license)

participated (driver_id, license, report_number, damage_amount)

Prob.2. What are the different types of entities and relationship in an education system ? Draw an E-R diagram. ***(R.G.P.V., Dec. 2010)***

Sol. The E-R diagram is shown in fig. 2.25.

MSInverter

Sol.

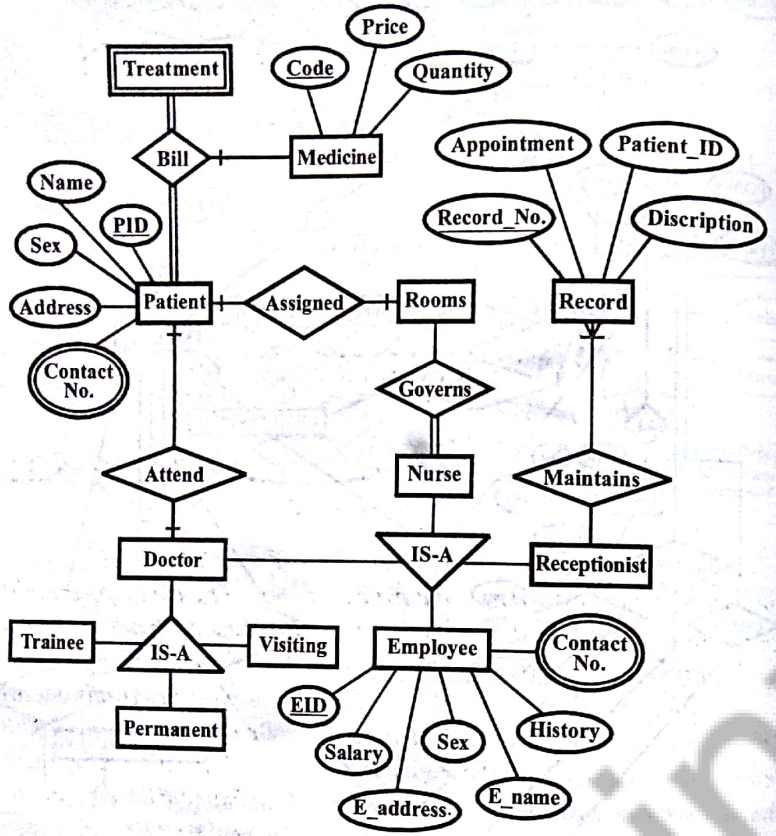


Fig. 2.27 E-R Diagram for Hospital Management System

Prob.6. Construct an E-R diagram for a hospital with a set of patients and set of medical doctors. Associate with each patient a log of the various test and examination conducted. ***(R.G.P.V., Dec. 2009)***

Or

Draw an E-R diagram for a hospital management system. Assume suitable entity and relations. ***(R.G.P.V., Dec. 2016)***

Sol. Fig. 2.28 shows an E-R diagram for the given problem. In the fig. 2.28, there are three entities patient, doctor, and test-examination. The entity patient has attributes p_id, p_name, age, weight, gender, address, illness. The entity doctor has attributes d_id, d_name, specialization, phone, address. The entity test_examination has attribute t_no, t_name, date, recommended_by, lab_name result.

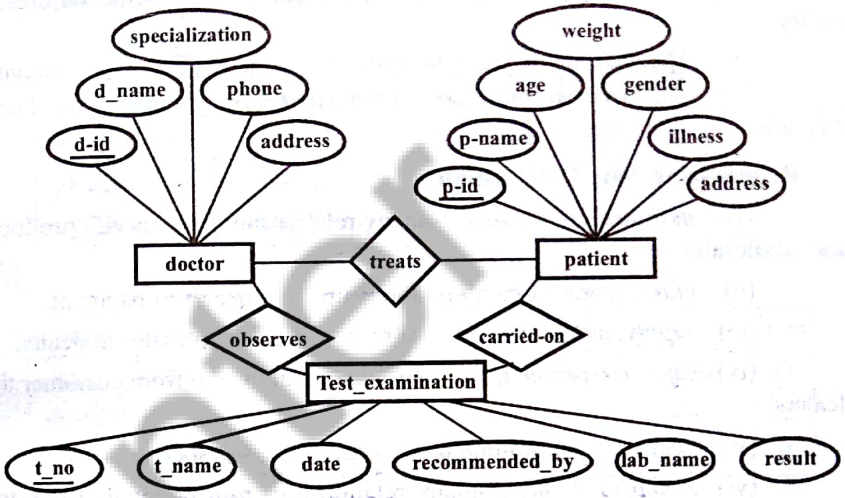


Fig. 2.28

Prob.7. Supreme products manufactures products like pressure cookers, cook-wares, water purifiers, food processors etc. The company markets its products to wholesalers all over the country and dealers sell them to customer. The company has five regional offices. Salespersons contact dealers and explain about products, incentives offered, training programs for whole salers and demonstrations for customers etc. Dealers place orders with the salespersons attached with the regional office of their location. After receiving goods they make payments, which may be in instalments. Company would like to develop a system to monitor sales of different products, performance of salespersons and orders from wholesalers.

Do the following –

(i) Identify_entities, attributes and relationships giving functionalities and draw and E-R diagram for the system.

(ii) Convert this to relational tables explaining the logic involved. ***(R.G.P.V., June 2008)***

Sol. (i) Identifying Entities, Attributes and Relationships, and E-R Diagram –

Entity Sets Designation – From the characteristics given in problem, we begin to identify entity sets and their attributes –

(i) The *product* entity set, with attributes p_id, p_name, price.

(ii) The *wholesaler* entity set, with attributes wholesaler_id, w_name, address, phone.

(iii) The *dealer* entity set, with attributes dealer_id, dname, address, phone.

MSinNepal

(iv) The *salesperson* entity set, with attributes s_id, sname, saddress, salary.

(v) The *order* entity set, with attributes o_number, o_date, o_amount.

(vi) The *payment* entity set, with attributes pay_number, pay_date, pay_amount, o_number.

Relationship Sets Designation –

(i) *assigned_to*, a many-to-many relationship set between product and wholesaler.

(ii) *make*, a one-to-many relationship from dealer to payment.

(iii) *supply*, a one-to-many relationship from wholesaler to dealer.

(iv) *supplied_product*, a one-to-many relationship from customer to dealers.

(v) *contact*, a one-to-one relationship from salesperson to dealers.

(vi) attach-to, a one-to-many relationship from regional_office to salesperson.

(vii) *contact*, a one-to-one relationship from dealers to salesperson.

(viii) *places*, a one-to-many relationship from dealers to orders.

E-R Diagram –

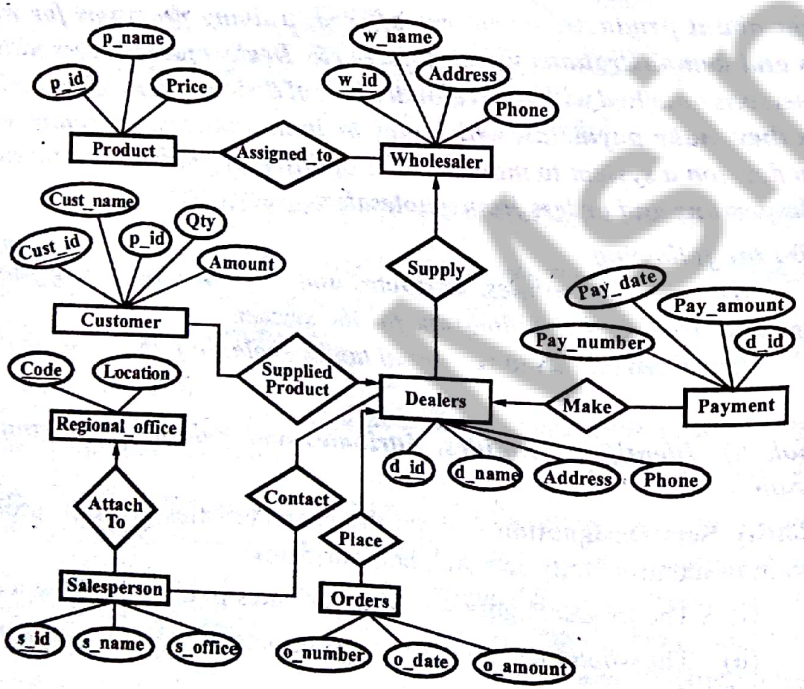


Fig. 2.29

(ii) Converting to Relational Tables –

Product

p_id	p_name	price
p1	cooker	200
p2	water purifier	500
p3	food processor	1000

Wholesaler

w_id	w_name	address	phone
w1	pavan	Indore	245397
w2	shyam	Ujjain	254390
w3	amit	Gwalior	23685

Dealers

d_id	d_name	address	phone
d1	Dilip	Gwalior	230695
d2	Kumar	Jabalpur	240545
d3	Santosh	Indore	260954

Payment

pay_number	pay_date	pay_amount	d_id
1	2 / 06 / 08	4000	d1
2	3 / 06 / 08	3000	d2
3	4 / 06 / 08	6000	d1

Customer

cust_id	cust_name	cust_address	p_id	Qty	amount
1	Sanjay	Gwalior	p1	1	200
2	Gopal	Ujjain	p3	1	1000
3	Amit	Indore	p2	1	500
4	Kishan	Jabalpur	p1	1	200

Orders

o_number	o_date	o_amount
1	2/05/08	4000
2	3/05/08	10000
3	3/05/08	6000

Salesperson

s_id	s_name	R_office
s1	Ajay Singh	r1
s2	Hemant Kumar	r2
s3	Nirmal	r3

Regional_office

Code	Location
R1	Gwalior
R2	Ujjain
R3	Indore
R4	Bhopal
R5	Jabalpur

Fig. 2.30

MSinverter

Prob.8. Draw E-R diagram for a small telecom marketing compo... database, assuming your own data requirements. (R.G.P.V., Dec. 201...

Sol.

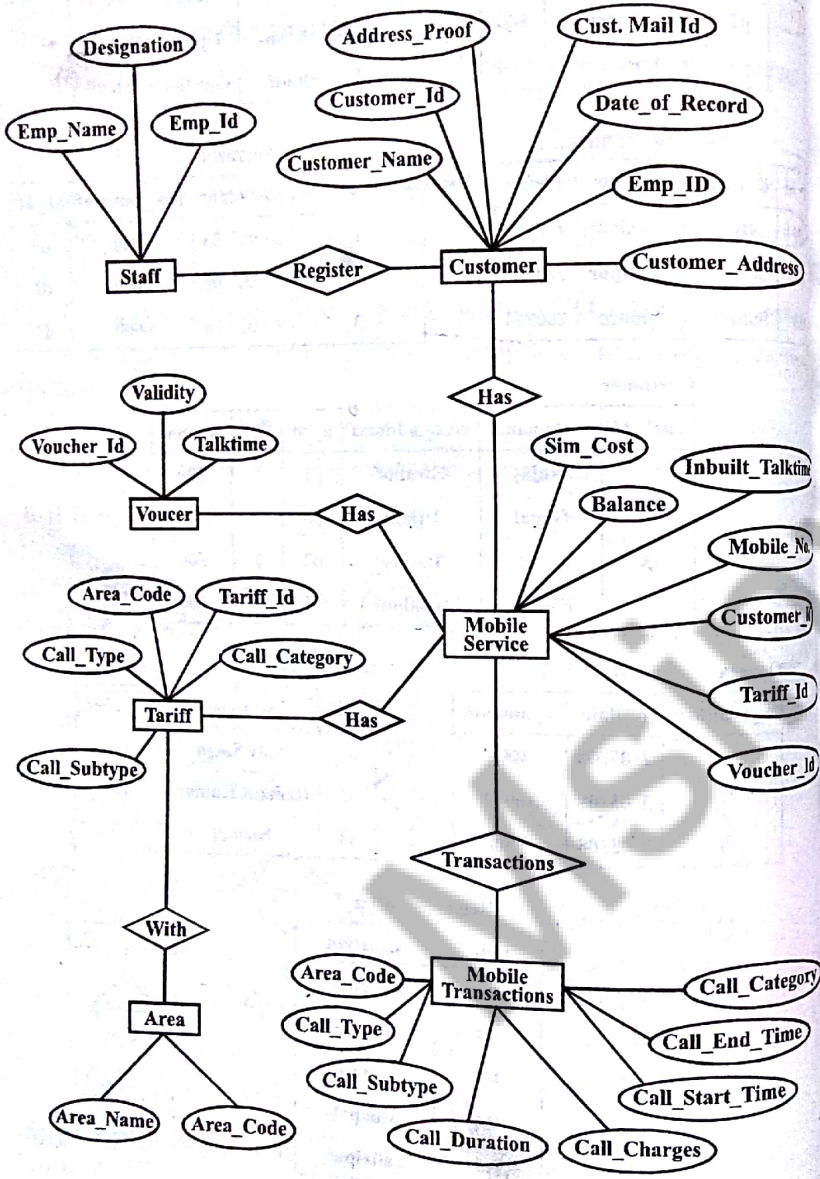


Fig. 2.31

Prob.9. (i) Consider ER diagram of BANK. List all entities in ER diagram.

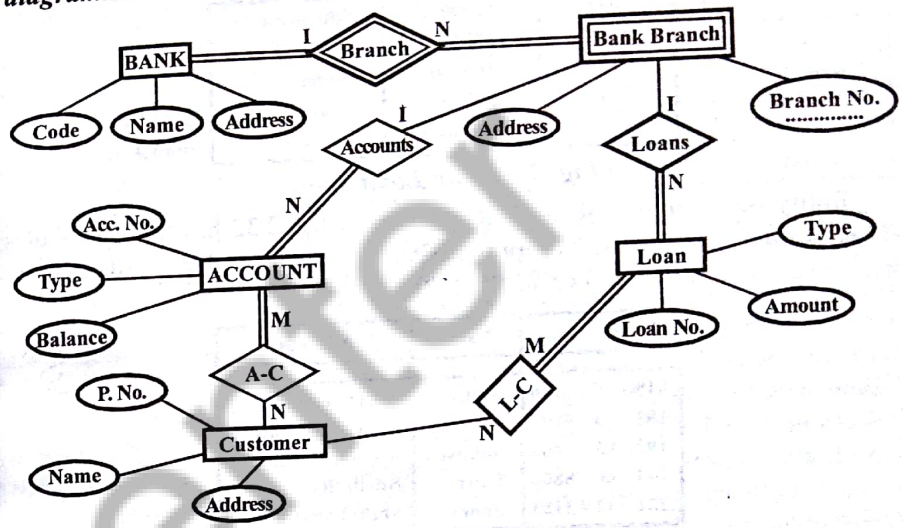


Fig. 2.32

(ii) List weak entities. Give its partial key and identifying relationship.

(iii) Design a database for above ER diagram (give tabular representation). ***(R.G.P.V., June 2007)***

Sol. (i) The entities are –

(a) BANK
(b) BANKBRANCH
(c) ACCOUNT
(d) CUSTOMER
(e) LOAN

(ii) The weak entity is BANKBRANCH

Identifying relationship is BRANCH

Partial key is Branch No. + code

(iii) Tabular Representation

(a) Tabular Representation of Strong Entity Sets –

Entity set ***loan*** of the E-R diagram in fig. 2.32 has three attributes ***Loan No.***, ***Amount*** and ***Type.*** We represent this entity set by a table called loan with three columns as shown in fig. 2.33.

MSInventer

Loan No.	Amount	Type
L-11	10000	Educational
L-14	20000	Personal
L-15	100000	Home
L-16	15000	Educational
L-17	200000	Home

Fig. 2.33 The Loan Table

Entity set ***customer*** of the E-R diagram in fig. 2.32 has three attribu ***P.No.***, ***Name,*** and ***address***. We represent this entity set by a table called custom with three columns as shown in fig. 2.34.

P. No.	Name	Address
019 - 28 - 3746	Smith	North, Rye
182 - 73 - 6091	Turner	Putnam, Stamford
192 - 83 - 7465	Johnson	Alma, Palo Alto
244 - 66 - 8800	Curry	North, Rye
321 - 12 - 3123	Jones	Main Harrison

Fig. 2.34 The Customer Table

Entity set ***account*** of the E-R diagram in fig. 2.32 has three attributes *Acc* ***No.***, ***Type*** and ***Balance***. We represent this entity set by a table called accoun with three columns as shown in fig. 2.35.

Acc. No.	Type	Balance
A - 101	saving	8000
A - 215	saving	15000
A - 102	saving	75000
A - 305	saving	8000
A - 201	saving	115000

Fig. 2.35 The Account Table

Entity set ***BANK*** of the E-R diagram in fig. 2.32 has three attributes code Name, and Address. We represent this entity set by a table called BANK wit three columns as shown in fig. 2.36.

Code	Name	Address
B - 01	ICICI	Rye
B - 02	HDFC	Stamford
B - 03	CITY BANK	Palo Alto
B - 04	MAX	Rye
B - 05	PNB	Harrison

Fig. 2.36 The BANK Table

(b) Tabular Representation of Weak Entity Sets – The entity set ***Bank Branch*** in the E-R diagram of fig. 2.32 has two attributes branch No. and Address. The primary key of the BANK entity set, on which ***Bank Branch*** depends, is code. Thus, we represent ***Bank Branch*** by a table with three columns labeled ***code***, Branch No. and Address as in fig. 2.37.

Code	Branch No.	Address
B - 01	A - 10	Rye
B - 02	A - 11	Stamford
B - 03	A - 12	Palo Alto
B - 04	A - 13	Rye
B - 05	A - 14	Harrison

Fig. 2.37 The Bank Branch Table

Prob.10. For the following problem definition – The musical company wants to store information about the musicians who perform on its albums. Each musician has a musician Id, a name, an address, and a phone number. Some musicians may have the same address and some of them may have more than one phone number. Each musician may play several instruments and an instrument may be played by several musicians. Each instrument has name and a musical key. The album recorded has a title, a copy right date, a format and an album identifier. Each album has a number of songs where a song has a title and an author. Each song may be performed by several musicians and a musician may perform a number of songs. One of the musicians of the song acts as a producer. A producer may produce several albums.

(i) Draw an E-R diagram.

(ii) Transform the E-R diagram to a relational schema.

(R.G.P.V., Dec. 2017)

Sol. (i) E-R Diagram –

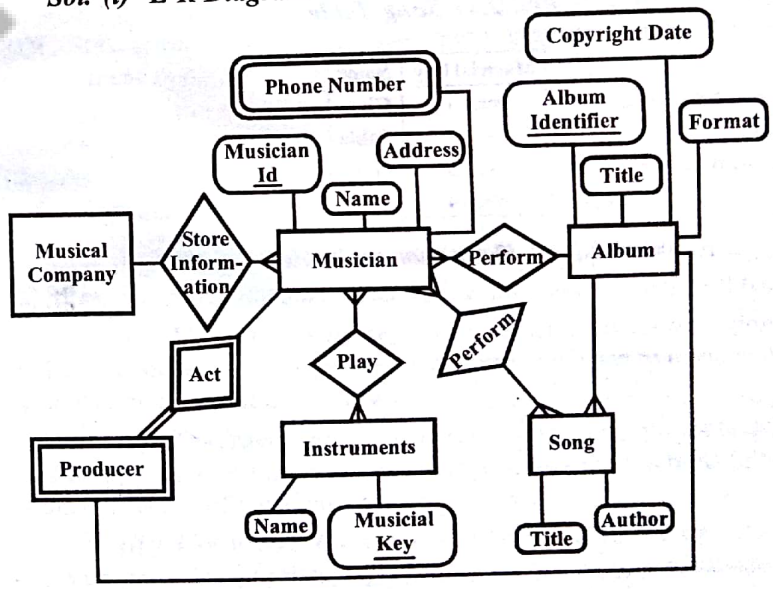


Fig. 2.38

(ii) Transform the E-R diagram to a relational schema –

Musician Id	Name	Address	Phone Number
m - 1	Jitu	KRG	4740012
m - 2	Virat	KRG	4740023
m - 3	Sonu	Kampoo	4740034/48
m - 4	Ravi	Morar	4740045/46

Fig. 2.39 Musician Table

Album Identifier	Title	Format	Copyright Date
a - 1	Robot	MPEG	2 - 1 - 19
a - 2	Zig Zag	MP4	3 - 1 - 19
a - 3	Love Triangle	AVI	31 - 12 - 18
a - 4	Sorry X	DIV X	20 - 2 - 19

Fig. 2.40 Album Table

Title	Author
try	Sonu
baby	Jitu
sorry	Virat
easy	Ravi

Fig. 2.41 Song Table

Musicial Key	Name
mk - 1	Gitar
mk - 2	Tabla
mk - 3	Piano
mk - 4	Flute

Fig. 2.42 Instrument Table

⌘⌘

UNIT 3

SQL – DATA DEFINITION IN SQL, UPDATE STATEMENTS AND VIEWS IN SQL, QUEL & QBE – DATA STORAGE AND DEFINITIONS, DATA RETRIEVAL QUERIES AND UPDATE STATEMENTS

Q.1. Define query language. ***(R.G.P.V., Dec. 2015)***

Ans. A DBMS provides a specialized language called the query language. Query languages are used for asking questions or queries that involve data in a database.

Q.2. What do you mean by the term SQL ?

Ans. SQL stands for structured query language. It is used to communicate with the database. Whenever any data is to be accessed from the database, we can do it through the SQL, and user can access the required data from the database, can manipulate data, can define the data in the database through SQL. SQL is supposed as structured query language for relational database management systems (RDBMS).

SQL uses the combination of relational algebra and relational calculus constructs. It has many other capabilities besides querying the database. It includes the features of defining the structure of the data, for modifying data in the database, and for specifying security constraints.

Q.3. Explain the basic structure of an SQL statement.

Ans. The basic structure of an SQL expression consists of three clauses –

(i) The select Clause – The select clause corresponds to the projection operation of the relational algebra. It is used to retrieve the desired attributes in the result of a query.

(ii) The from Clause – The from clause corresponds to the cartesian product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.

(iii) The where Clause – The where clause corresponds to the selection predicate of relational algebra. It consists of a predicate involving attributes of the relations that appear in the ***from*** clause.

Msinghar

A typical SQL query has the form –

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
where P

Here, each A_i denotes an attribute, each r_i denotes a relation and P is condition or predicate.

Q.4. Discuss the various aggregate functions in SQL with an example for each. ***(R.G.P.V., Dec. 20***

Or

What is aggregate function ? Explain any five aggregate functions with example. ***(R.G.P.V., May 20***

Or

Explain any four aggregate functions of SQL with an example for each. ***(R.G.P.V., Dec. 20***

Ans. Aggregate functions are functions that take a collection of values as input and return a single value. In SQL, we have five aggregate functions -

(i) Average : avg (ii) Minimum : min
(iii) Maximum : max (iv) Count : count
(v) Total : sum.

Here, sum and avg functions work only for numbers, while min, max and count work equivalently well for alphabets i.e., string.

(i) Average : avg – This function takes as input, a collection of numbers and returns a single number, which is average of all the numbers. For example, suppose we want to find average of amount of orders in order table, then query can be written as –

select avg(amt)
from orders

This query will result a single attribute named avg(amt) having only one value that is average of all amounts. If we wish to get avg with heading of field as average then query should be written as follows –

select avg(amt) average
from orders

Now, we will have only one attribute with heading average and only one value that is average of amount from order field.

We can also use group by clause. The group by clause is useful in circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples.

For example,

select snum, avg(amt)
from orders
group by snum

This query will result average amount for each snum.

We can also use having clause in SQL, for applying some other restrictions. For example,

select snum, avg(amt)
from orders
group by snum
having avg(amt) > 1200

The predicates in the 'having' clause are applied after the formation of groups, so the aggregate functions may be used. This query will give us only snum which contains average amount more than 1200. The 'having' clause is used only with the aggregate functions.

(ii) Minimum : min – This function is used to find minimum number from a group of numbers. For example, suppose we want to find the amount with minimum value from orders table –

select min(amt)
from orders

This query will give us with the minimum amount i.e., 18.69.

(iii) Maximum : max – This function is used to find maximum number from a given group of numbers. For example, suppose we wish to find maximum amount from orders table –

select max(amt)
from orders

This query will result with only one attribute that contains only one value i.e., the maximum amount in order table. In our example is 9891.88.

(iv) Count : count – This function is used to return total number of values in a given attribute, means total count. For example, if we want to find total number of snum in orders table then

select count(snum)
from orders

Here, count counts the number of snum in the table and displays the total number of snum. In our example, this value will be 10. But, here some of the snum are repeating. For example, 1001, 1002, 1003, etc. So, if we wish to count one snum only once, then we can use distinct in the following way –

select count(distinct snum)
from orders

In our example, this value is 5.

If we wish to find total orders in orders table, then we can also write as –

select count(*)
from orders

This will result a value 10.

(v) Total : sum – This function is used to return the sum of numbers. This function just adds all the numbers given to it as input and

return the sum of these numbers. For example, if we want to find the sum amounts of orders from orders table then –

```
select sum(amt)
from orders
```

If we want to find the sum of amount for each snum separately, then have to write as

```
select snum, sum(amt)
from orders
group by snum
```

This will give us snum with total of amounts for each snum separately. a 'where' clause and a 'having' clause appear in the same query, the predica in 'where' clause is applied first. The tuples satisfying the 'where' predica are then placed into groups by the group by clause. The 'having' clause present is then applied to each group, the groups that do not satisfy the havin clause predicate are removed. The remaining groups are used by the sele clause to generate the tuples of the result of the query.

Q.5. What are the various characteristics of SQL ? Discuss five aggrega functions with a suitable example. ***(R.G.P.V., Dec. 201***

Ans. **Characteristics of SQL** – The main characteristics of the SQ language is that it is a declarative or nonprocedural language. This implies tha the programmer does not need to specify step-by-step all the operations tha the computer needs to carry out to obtain a particular result. Instead, th programmer indicates to the DBMS what needs to be accomplished and the lets the system decide on its own how to obtain the desired result. Th statements or commands that compose the SQL language are generally divide into two major categories or data sublanguages. Each sublanguage is concerne with a particular aspect of the language. One of these sublanguages known a the data definition language or DDL and the other is known as data manipulatio language or DML. The SQL language can be used interactively or in it embedded form.

Five Aggregate Functions with Suitable Example – Refer to Q.4.

Q.6. What are the different parts of SQL ?

Ans. The different parts of SQL language are as follows –

(i) Data Definition Language – It provides the commands fo defining relation schemas, deleting relations, creating indices and modifyin relation schemas.

(ii) Data Manipulation Language – It includes a query languag based on both the relational algebra and the tuple relational calculus. It also provides commands to insert, delete, and modify tuples in the database.

(iii) View Definition – It includes commands for defining views.

(iv) Transaction Control – It includes commands for specifying the beginning and ending of transactions.

(v) Integrity – The SQL DDL provides commands for specifying integrity constraints, that the data stored in the database must satisfy. Updates that can violate integrity constraints are disallowed.

(vi) Authorization – The SQL DDL provides commands for specifying access rights to relations and views.

(vii) Embedded SQL and Dynamic SQL – Embedded and dynamic SQL define how SQL statements are embedded within programming languages such as C, C++, Java.

Q.7. What are the relational algebra operations supported in SQL ? Write the SQL statements for each operation. ***(R.G.P.V., June 2011, Dec. 2012)***

Or

Explain about set operation in relational algebra. (R.G.P.V., Dec. 2016)

Ans. **Set Operations** – The SQL operations union, intersect, and except operate on relations and correspond to the relational algebra operations ∪, ∩ and –

(i) The Union Operation – To find all customers having a loan, an account, or both at the bank, we write

```
(select customer-name from depositor)
union
(select customer-name from borrower)
```

The union operation automatically eliminates duplicates, unlike the select clause.

(ii) The Intersect Operation – To find all customers who have both a loan and an account at the bank, we write

```
(select distinct customer-name from depositor)
intersect
(select distinct customer-name from borrower)
```

The intersect operation automatically eliminates duplicates.

(iii) The Except Operation – To find all customers who have an account but no loan at the bank, we write

```
(select distinct customer-name from depositor)
except
(select customer-name from borrower)
```

The except operation automatically eliminates duplicates.

Q.8. Explain the EXISTS and UNIQUE functions of SQL. Give an example for each. ***(R.G.P.V., June 2008)***

Ans. The EXISTS function in SQL is used to check whether the result of a correlated nested query is empty (i.e., contains no tuples) or not. The use of

EXISTS is illustrated with an example. Suppose that the query is "Retrieve name of each employee who has a dependent" with the same first name same sex as the employee. This query is formulated as follows –

```
select E.FNAME, E.LNAME from EMPLOYEE as E where exists
    (select * from DEPENDENT where E.SSN = ESSN AND E.SE
        = SEX AND E.FNAME = DEPENDENT_NAME);
```

EXISTS and NOT EXISTS are usually used in conjunction with a correla nested query. In this query, the nested query references the SSN, FNAM and SEX attributes of the EMPLOYEE tuple, evaluate the nested query, whi retrieves all DEPENDENT tuples with the same social security number, se and name as the EMPLOYEE tuple; if at least one tuple EXISTS in the res of the nested query, then select that EMPLOYEE tuple. In general, EXISTS(returns TRUE if there is *at least one tuple* in the result of query Q, and returns FALSE otherwise. On the other hand, NOT EXISTS(Q) returns TRU if there are no tuples in the result of query Q, and it returns FALSE otherwis To illustrate the use of NOT EXISTS, we can write the query "Retrieve th names of employees who have no dependents", as follows –

```
select FNAME, LNAME
FROM EMPLOYEE
where not exists (select*from DEPENDENT where SSN = ESSN)
```

The description of this query is as follows – for each EMPLOYEE tupl the correlated nested query selects all DEPENDENT tuples whose ESSN valu matches the EMPLOYEE SSN; if the result is empty, no dependents are relate to the employee, so we select that EMPLOYEE tuple and retrieve its FNAM and LNAME.

There is another SQL function UNIQUE (Q) that returns true if there ar no duplicate tuples in the result of query Q; otherwise, it returns false. Usin the ***unique*** construct, we can write the query. "Find all customers who hav at most one account at the Perryridge branch" as follows –

```
select T.customer_name from depositor as T
    where unique (select R.customer_name
        from account, depositor as R
        where T.customer_name = R.customer_name
        and R.account_number = account.account_number
        and account.branch_name = 'Perryridge')
```

We can test for the existence of duplicate tuples in a subquery by usin the *not unique* construct. Using the *not unique* construct, we can write th query "Find all customers who have at least two accounts at the Perryridg branch", as follows –

```
select distinct T.customer_name
from depositor T
where not unique(select R.customer_name
    from account, depositor as R
    where T.customer_name = R.customer_name
    and R.account_number = account.account_number
    and account.branch_name = 'Perryridge')
```


Q.9. How is the modification of database done in SQL ?

Ans. Along with the extraction of the information from the database, the modifications to the database is possible through SQL such as add, remove or change the information using SQL commands.

(i) Insertion – To insert the data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. The attribute values for inserting tuples must be the members of the attributes domain. Similarly, the tuples inserting must be of correct arity. The simplest insert statement is a request to insert one tuple. For example,

```
insert into orders
values (3012, 456.32, 11/05/99, 2006,1001)
```

This query will insert a tuple with onum 3012, amt 456.32, odate 11/05/99, cnum 2006, snum 1001 into the orders relation.

If we do not want to insert the values in all attributes, then we should specify column name in the following way –

```
insert into orders (onum, amt, snum)
values (3012, 456.32, 1001)
```

In this case, the remaining fields, i.e., odate, cnum store null values. For the character value single quotes should be used. For example,

```
insert into customer (cnum, cname, city)
values (2009, 'Priya', 'India')
```

If we want to insert more than one tuples one after other, then the query should be used in the following way –

```
insert into customer (cnum, cname, city)
values (&cnum, '&cname', '&city')
```

In above way, first of all cnum is asked, then cname and then city is asked from the user. So that the insertion of more than one tuple in a relation becomes easy.

(ii) Deletion – A delete request is expressed in much the same way as a query. We can delete only whole tuples, we cannot delete values on only particular attributes. In SQL, the deletion is expressed by

```
delete from r
where p
```

where r is a relation and p is a predicate.

The delete statement first finds all the tuples t in r for which predica p(t) is true and then deletes them from r. For example,

```
delete from orders
```

This query deletes all tuples from relation orders. We can also apply son conditions in delete command after where clause.

```
delete from orders
where onum = 3001
```

The above query will delete only the tuple of which order number 300 The delete command operates on only one relation. If we want to delete th tuples from several relations, we must use one delete command for eac relation. The predicate in the 'where' clause may be as complex as sele command's 'where'clause. For example,

```
delete from orders
where cnum = any (select cnum from customer where cname = 'Hoffman'
```

This query will delete orders from orders table, whose cnum is equal to an of the cnum that are present in customer table whose customer name is Hoffma

(iii) Updates – Sometimes, we want to change a value in a tupl without changing all the values in the tuple. For this purpose, the updat statement can be used. For example, if we want to increase the commissio of sales person in salespeople relation by 5%, then the following updatin should be done.

```
update salespeople
set comm = comm * 1.05
```

In above query, we can also apply the condition, i.e., increase th commission of salesperson by 6 percent, only for those who have commissio greater than 0.13, and for others increase the commission only by 4 percent This could be done by writing two queries.

```
update salespeople
set comm = comm * 1.06
where comm > 0.13
update salespeople
set comm = comm * 1.04
where comm < = 0.13.
```

Q.10. Explain following command with an example
(i) Insert (ii) Group by

Ans.(i) Insert – Refer to Q.9. ***(R.G.P.V., June 2016***

(ii) Group By – The group by clause is part of SELECT statement This clause groups rows based on distinct values that exist for specified columns

For example,

```
SELECT product_no, sum (qty_ordered)
FROM sales_order_details
GROUP BY product_no;
```

Here, the common rows in column product_no are grouped together and total quantity ordered for each product is displayed on screen.

Q.11. Write short note on missing information. (R.G.P.V., Dec. 2014)

Ans. Information is often missing in the real world. For example, date of birth, address, and so on are common and familiar to all of us. Thus, there is a need to deal with such missing information inside our database systems. And the approach to this problem most commonly adopted in practice – in SQL in particular, and hence in most commercial products – is based on nulls and three-valued logic. For example, we might not know the weight of some part (P). So we might say, loosely, that the weight of the part is NULL meaning, more precisely, that we do know the part exists, and of course we also know it has a weight, but to repeat, we do not know what that weight is. The concept of nulls, at least as that term is usually understood, inevitably leads us into a logic in which there are three truth values – true, false and unknown.

Q.12. What are views ? Why they can't be used for updates ?
(R.G.P.V., June 2014)

Or

What is a view in SQL ? ***(R.G.P.V., Dec. 2015)***

Ans. A view in SQL terminology is a single-table that is derived from other tables. These other tables could be base tables or previously defined views. A view does not necessarily exist in physical form. It is considered a virtual table, in contrast to base tables whose tuples are actually stored in the database. This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view.

Q.13. With relevant examples discuss the following in SQL.
(i) Data definition language (ii) Data manipulation language
(iii) Data control language (iv) Views.
(R.G.P.V., Dec. 2012)

Ans. (i) Data Definition Language (DDL) – A database schema is defined by a set of definitions expressed by a special language called ***data definition language*** **(DDL)**. After compilation of DDL statements, we get a set of tables that is stored in a special file called ***data dictionary*** or ***data directory***.

(ii) Data Manipulation Language (DML) – A data manipulation language (DML) is a language that enables users to access or manipulate data as organized by appropriate data model. We have basically two types of languages.

MsinVenkat

(a) Procedural DMLs – This specifies, **what** data are needed and **how** to get those data.

(b) Non-procedural DMLs – This specifies, **what** data are needed without specifying how to get those data.

(iii) Data Control Language (DCL) – We must know how to give access to and revoke accesses from various tables to different users. This is very important in case of a multi-user database system. This can be achieved with the use of Data Control Language (DCL). Suppose there are two persons working in the personnel department – Hari and Mohan. Only they should have access to the *Employee* table. We can secure access by using the GRANT and REVOKE statements provided by SQL as follows. First, we remove all accesses to the table by everybody.

```
REVOKE ALL ON Employee
FROM PUBLIC
```

Here, ALL means the ability to perform any DML operation such as SELECT, INSERT, UPDATE, or DELETE. Thus, we do not allow anyone (because we specify PUBLIC) to perform any operation on the *Employee* table. In other words, PUBLIC means *everybody.* Now, we shall allow only Hari and Mohan the necessary authority to access then *Employee* table.

```
GRANT ALL
ON Employee
TO Hari, Mohan
```

This would allow Hari and Mohan to perform any DML operation on the *Employee* table. Suppose at some stage, Mohan is on leave and a third employee, David, temporarily takes his place. We need to provide only SELECT access to David on the *Employee* table. This can be done as follows –

```
GRANT SELECT
ON Employee
TO David
```

David cannot update the information in the *Employee* table in any manner. He can only see what it contains.

(iv) Views – Refer to Q.12.

Q.14. What is view ? How to create a view ?

Ans. **View** – Refer to Q.12.

For creating views in SQL, we use the create view command. For defining a view, we must give a name to view and must state the query that computes the view.

The form of create view command is –

create view viewname as <query expression>

For example suppose, we have a relation emp with fields emp_no, ename, salary, designation, department_no, date_of_joining.

We create a view on this relation, with fields emp_no, ename, sal, bonus under the viewname emp_bonus.

```
create view emp_bonus
(emp_no, ename, sal, bonus)
as select emp_no, ename, salary, salary * 0.4
from emp;
```

In the above example, we will have bonus field that will be computed by multiplying 0.4 to the salary field.

There are some other examples, in which we can also use union to create a view. For example, consider the view consisting of branch names and the names of customers who have either an account or a loan at that branch. We give the name of this view customer.

We define the view as follows –

```
create view customer as
(select branch_name, customer_name
from depositor, account
where depositor.account_number = account.account_ number)
union
(select branch_name, customer_name
from borrower, loan
where borrower.loan_number = loan.loan_number)
```

The view names may appear in any place where a relation may appear.

For example, suppose we want to access salary and bonus field from view emp_bonus for employee 'Ashish'.

Table 3.1 The emp Relation

emp_ no	*ename*	*salary*	*designation*	*DOJ*	*depart_ no*
1	Piyush	10,000	Analyst	05 / 10 / 2000	1001
2	Ravi	8,000	Clerk	01 / 01 / 2001	1021
3	Sushil	20,000	Manager	10 / 02 / 1990	1022
4	Ashish	25,000	Manager	11 / 04 / 1992	1001
5	Khushi	15,000	Engineer	07 / 08 / 1995	1030

Then, we will write the following expression –

```
select sal, bonus
from emp_bonus
where ename = 'Ashish'
```

Similarly, we can find customers of the branch "Delhi" by writing

```
select customer_name
from customer
where branch_ name = "Delhi"
```

MSIPU Paper

For dropping a view, we have the following simple expression –

```
drop view viewname
```

For example,

```
drop view emp_bonus
```

or

```
drop view customer
```

Q.15. Define the following term in context of SQL –

(i) Views

(ii) Join

(ii) In.

(R.G.P.V., Dec. 20

Ans. (i) Views – Refer to Q.12.

(ii) Join – Refer to Q.31, Unit-II.

(ii) In – IN operator uses a value list. All of the values in the l must be of the same data type. Each of the values in the value list is compa to the attribute.

For example –

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN (21344, 24288);
```

Here, if the V_CODE value matches any of the values in the list, the r is selected. In this example, the rows selected will be only those in which t V_CODE is either 21344 or 24288.

If the attribute used is of a character data type, the list values must enclosed in single quotation marks. For instance, if the V_CODE had be defined as CHAR(5) during the table-creation process, the preceding que would have read,

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN ('21344', '24288');
```

The IN operator is especially valuable when it is used in conjunction wi subqueries. For example, suppose you want to list the V_CODE and V_NAM of only those vendors who provide product. In that case, you could use subquery within the IN operator to automatically generate the value list. T query would be

```
SELECT      V_CODE, V_NAME
FROM        VENDOR
WHERE       V_CODE IN (SELECT V_CODE
                       FROM PRODUCT);
```


Q.16. What is the significance of the views ? Explain the different Codd's rule. ***(R.G.P.V., June 2009)***

Or

Explain the different CODD rules. ***(R.G.P.V., Dec. 2010)***

Or

Specify CODD's norms. ***(R.G.P.V., June 2014)***

Ans. **Views** – Refer to Q.12.

Codd's Relational Database Rules – Dr. E.F. Codd published a list of 12 rules to define a relational database system as given in table 3.2 –

Table 3.2 Dr. Codd's 12 Relational Database Rules

Rule	*Rule Name*	*Description*
1	Information	All information in a relational database must be logically represented as column values in rows within tables.
2	Guaranteed access	Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.
3	Systematic Treatment of nulls	Nulls must be represented and treated in a systematic way, independent of data type.
4	Dynamic on-line catalog based on the relational Model	The metadata must be stored and managed as ordinary data, that is, in tables within the database. Such data must be available to authorized users using the standard database relational language.
5	Comprehensive data sublanguage	The relational database may support many languages. However it must support one well-defined declarative language with support for data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization and transaction management (begin, commit, and rollback).
6	View updating	Any view that is theoritically updatable must be updatable through the system.
7	High-level insert, update and delete	The database must support set-level inserts, updates and deletes.
8	Physical data independence	Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.

Msirvenier

9	Logical data independence	Application programs and ad hoc facilities a logically unaffected when changes are mad to the table structures that preserve the orig nal table values (changing order of colum or inserting columns).
10	Integrity independence	All relational integrity constraints must be d finable in the relational language and stored in th system catalog, not at the application level.
11	Distribution independence	The end users and application programs ar unaware and unaffected by the data locatio (distributed vs local database).
12	Nonsubversion	If the system supports low-level access to the data, there must not be a way to bypas the integrity rules of the local databases).

Note – Rule zero – All preceding rules are based on the notion that order for a database to be considered relational, it must use its relational facilit exclusively to manage the database.

Q.17. What is NULL ? Give an example to illustrate testing for NUL in SQL. ***(R.G.P.V., June 201***

Ans. In the SQL allow us the use of null value to represent the absence information about to the value of an attribute. The keyword NULL is used f a predicate to check for a null value. Hence, to find all product numbers th can appear in the product relation with the null values for amount.

So, we can write query here

Select product_number from product

Where amount in null;

If we checks for the absence of a null value so the predicate cannot b null. Because the NULL value is used for arithmetic and comparison operatio then occurred several complications.

The result of an arithmetic expression is null if any of the input values i null. SQL treats as unknown the result of any comparison involving a nu value.

Q.18. Describe NULL value and dangling tuple problems.

(R.G.P.V., Dec. 2006, June 201

Or

Write short note on problems with dangling tuple. (R.G.P.V., Dec. 201

Ans. The problems associated with nulls must be carefully considered while designing a relational database schema. There is no fully satisfactory relational design theory as yet that includes null values. One problem occurs when some tuples have null values for attributes that will be used to JOIN individual relations in the decomposition. To illustrate this, consider the database shown in fig. 3.1 (a), where two relations EMPLOYEE and DEPARTMENT are shown. The last two employee tuples – Berger and Benitez – represent newly hired employees who have not yet been assigned to a department. Now suppose that we want to retrieve a list of (ENAME, DNAME) values for all the employees. If NATURAL JOIN operation is applied on EMPLOYEE and DEPARTMENT [fig. 3.1 (b)], the two aforementioned tuples will not appear in the result. The OUTER JOIN operation can deal with this problem. If we take the LEFT OUTER JOIN of EMPLOYEE with DEPARTMENT, tuples in EMPLOYEE that have null for the join attribute will still appear in the result, joined with an "imaginary" tuple in DEPARTMENT that has nulls for all its attribute values. Fig. 3.1 (c) shows the result.

EMPLOYEE

ENAME	SSN	BDATE	ADDRESS	DNUM
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	null
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	null

DEPARTMENT

DNAME	DNUM	DMGRSSN
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

(a) A Database with Nulls for Some Join Attributes

ENAME	SSN	BDATE	ADDRESS	DNUM	DNAME	DMGRSSN
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

(b) Result of Applying NATURAL JOIN to the EMPLOYEE and DEPARTMENT Relations

MSRGPV

ENAME	SSN	BDATE	ADDRESS	DNUM	DNAME	DMGRSSN
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	null	null	null
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	null	null	null

(c) Result of Applying OUTER JOIN to EMPLOYEE and DEPARTMENT

Fig. 3.1 Illustrating Null Value Join Issues

In general, whenever a relational database schema is designed where two or more relations are interrelated via foreign keys, particular care must be devoted to watching for potential null values in foreign keys. This can cause unexpected loss of information in queries that involve joins on that foreign key.

A related problem is that of *dangling tuples,* which may occur if we carry a decomposition too far. Suppose that we decompose the EMPLOYEE relation of fig. 3.1 (a) further into EMPLOYEE_1 and EMPLOYEE_2, shown in fig 3.2 (a) and (b). If we apply the NATURAL JOIN operation to EMPLOYEE_1 and EMPLOYEE_2 we obtain the original EMPLOYEE relation. However, we may use the alternative representation shown in fig. 3.2 (c), where we do not include a tuple in EMPLOYEE_3 if the employee has not been assigned a department. If we use EMPLOYEE_3 instead of EMPLOYEE_2 and apply a NATURAL JOIN on EMPLOYEE_1 and EMPLOYEE_3, the tuples for Berger and Benitez will not appear in the result. These are called ***dangling tuples*** because they are represented in only one of the two relation that represent employees and hence are lost if we apply an (inner) join operation.

EMPLOYEE_1

ENAME	SSN	BDATE	ADDRESS
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX

(a) The Relation EMPLOYEE_1
(Includes All Attributes of EMPLOYEE Except DNUMBER)

EMPLOYEE_2

SSN	DNUM
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	null
888664444	null

(b) The Relation EMPLOYEE_2 (Includes DNUMBER Attribute with Null Values)

EMPLOYEE_3

SSN	DNUM
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

(c) The Relation EMPLOYEE_3 (Includes DNUMBER Attribute but does not include Tuples for which DNUMBER has Null Values)

Fig. 3.2 The "Dangling Tuple" Problem

Q.19. List the SQL statements used for transaction control.
(R.G.P.V., June 2015)

Ans. A transaction is an atomic unit of work that is either completed in its entirety or not done at all. Transaction control statements are as follows–

(i) BEGIN_TRANSACTION – This marks the beginning of transaction execution.

(ii) READ or WRITE – These specify read or write operations on the database items that are executed as part of a transaction.

(iii) END_TRANSACTION – This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability or for some other reasons.

(iv) COMMIT_TRANSACTION – This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.

(v) ROLLBACK (or ABORT) – This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Q.20. Write short note on the following –
(i) Data manipulation language (DML)
(ii) Data definition language (DDL)
(iii) Transaction control statement (TCS)
(iv) Data control language (DCL).
(R.G.P.V., Dec. 2013)

MSWriter

Ans. ***(i) DML*** – Refer to Q.13 (ii).

(ii) DDL – Refer to Q.13 (i).

(iii) TCS – Refer to Q.19.

(iv) DCL – Refer to Q.13 (iii).

Q.21. Describe the features of embedded SQL. (R.G.P.V., Dec. 2014)

Ans. Some basic features of the embedded SQL are as follows –

(i) The embedded SQL statements appear in the midst of the host programming language. It usually does not matter whether the SQL statements are written in upper-case or lower-case. Usually the style of the host language is followed.

(ii) Embedded SQL statements are prefixed by a delimiter – EXECSQL – so that they can be distinguished from the host language statements.

(iii) If an embedded SQL statement extends over multiple lines, the host language strategy for statement continuation is used.

(iv) Every embedded SQL statement is terminated with a delimiter.

(v) Every embedded SQL program should include a host variable.

Q.22. What is QUEL and how data retrieval is done in QUEL ? Explain.

Ans. QUEL is query language of INGRES, a relational DBMS developed at the University of California, Berkeley, to run under UNIX operating system. The language is used either in stand-alone manner, by typing commands to the QUEL processor or embedded in C programming language. QUEL queries mainly use three clauses as follows –

(i) range of – Each tuple variable is declared in a range of clause.

(ii) retrieve – The retrieve clause is similar in function to the select clause of SQL.

(iii) where – The where clause has the selection predicate.

A simple query is written in the form

range of t_1 is r_1
range of t_2 is r_2

---
range of t_m is r_m
retrieve $(t_{i1}, A_{j1}, t_{i2}, A_{j2}, \ldots\ldots, t_{in}, A_{jn})$
where P ??? to author.

Here, t_i denotes a tuple variable,
r_i denotes a relation, and
A_{jk} is an attribute.

Relational algebra operations, such as intersection, union and minus cannot be applied in QUEL. The nested queries are not allowed i.e., we cannot have nested retrieve where clause inside a where clause. For example, if we want to find all the supplier name from salespeople relation. We write

```
range of t is salespeople
retrieve(t.sname)
```

This query results all the duplicate too. So, if we want one value to be resulted once only we can use unique operator in following way –

```
range of t is salespeople
retrieve unique (t.sname)
```

This query will eliminate all the duplicate values.

Now for applying condition, suppose we want to find all customer names who live in city Berlin and have rating greater than 100. We can write as

```
range of t is customer
retrieve unique (t.cname)
where city = 'Berlin' and rating > 100
```

QUEL also allows to apply the queries on more than one relation.

For example, if we want to find the names of customers whose suppliers live in London. We write as

```
range of t is customer
range of s is salespeople
retrieve unique (t.cname)
where s.city = 'London'
and s.snum = t.snum
```

Q.23. How the modification of database can be done in QUEL? Explain. (R.G.P.V., June 2016)

Ans. The modifications of database in QUEL is similar to modifications in SQL, although the syntax differs slightly.

(i) Deletion – QUEL allows the deletion of tuples from a relation. This can be done in following way –

```
range of t is r
delete t
where p
```

Here, t is a tuple variable defined implicitly, r is a relation and p is a predicate, can be any valid QUEL predicate. For example, if we want to delete all the tuples from relation orders. We can write

```
range of t is orders
delete t
```

If we do not want to delete all tuples, but only some, for this we have to apply some condition. For example, "if we want to delete all the information

MSI Meerut

of salespersonRifkin". For this we write,

```
range of t is salespeople
delete t
where t.sname = "Rifkin"
```

Delete operation can be applied on more than one relation. For example, "if we want to delete all the orders made by salesperson Serres". We write

```
range of t is orders
range of s is salespeople
delete t
where t.snum = s.snum and
s.sname = "Serres"
```

(ii) Insertion – For insertion, we take the keyword append. Insertion in QUEL takes two general forms : insertion of a single tuple, and insertion of a set of tuples. For example, "if we want to insert a tuple (1008, Priya, France,.09) in salespeople relation". Then we write,

```
append to salespeople (snum = 1008,
sname = "Priya",
city = "France",
comm = 0.9)
```

(iii) Updates – QUEL updation can be done using command replace. For example, "if we want to increase commission of each salespeople by 3 percent". Then we write,

```
range of t is salespeople
replace t (comm = 1.03*t.comm)
```

Also, we can apply conditions during updation in QUEL. For example, "if we wish to increase the commission by 5 percent of salespeople who have commission greater than 0.11 and by 3 percent for all other". We can write

```
range of t is salespeople
replace t(comm = 1.05*comm)
where t.comm > 0.11
replace t(comm = 1.03*comm)
where t.comm ≤ 0.11
```

Thus, the updation in QUEL is also similar to SQL.

Q.24. How can the aggregate functions be applied in QUEL ? Explain.

Ans. QUEL uses the aggregate functions sum, avg, count, min, and max. The argument of such a function can be any expression involving components of a single relation, constants, and arithmetic operators. The components must all be referred to as t. A, for some one tuple variable t and various attributes A. For example, "if we want the net amount of all orders in orders relation", we would write,

```
range of t is orders
retrieve (sum (t.amt))
```


The functions countu, sumu and avgu are also used in QUEL. The functions countu, sumu and avgu are identical to count, sum, and avg respectively, except that they remove duplicates from their operands.

Aggregate functions in Quel compute functions on group of tuples, but in a different way from SQL. QUEL aggregate expressions can take the following forms –

```
aggregate function (t.A)
aggregate function (t.A where P)
aggregate function (t.A by S.B1, S.B2 ,....., S.Bn, where P)
```

where aggregate function is one of count, sum, avg, max, min, countu, sumu, avgu, or any t and s are tuple variables.

A, B_1, B_2 .B_n, are attributes. P is a predicate, similar to the where clause in a retrieve.

First form of aggregate function i.e., aggregate function (t.A) is shown in previous example.

Now, the second form of aggregate function (t.A where P) can be shown as below –

"Let we wish to find the average amount orders for all orders made by supplier number 1001".

```
For this we write,
range of t is orders
retrieve avg (t.amt where t.snum = 1001)
```

Aggregate operations can also be applied in where clause as follows –

"Let we wish to find all amounts that have an amount higher than the average of all amounts in the order relation".

```
For this we write,
range of u is orders
range of t is orders
retrieve (t.amt)
where t.amt > avg (u.amt)
```

Maximum, minimum, and count functions can also be used in similar manner.

For example, "if we wish to find the maximum amount order in orders relation", we can write –

```
range of t is orders
retrieve max (t.amt)
```

For finding minimum amount order in orders relation. We can write –

```
range of t is orders
retrieve min (t.amt)
```

Similarly, for a counting total number of orders in orders relation. We can write –

```
range of t is orders
retrieve count (t.amt)
```

MsinVenter

Q.25. Write short note on QBE. ***(R.G.P.V., Dec. 2006)***

Ans. The Query-By-Example (QBE) language is important because it is one of the first graphical query languages with minimum syntax developed for database systems. It was developed at IBM research and is available as an IBM commercial product as part of the QMF (Query Management Facility). It differs from SQL in that the user does not have to specify a structured query explicitly; rather the query is formulated by filling in templates of relations that are displayed on a monitor screen. The user does not have to remember the names of attributes or relations, because they are displayed as part of these templates. In addition, the user does not have to follow any rigid syntax rules for query specification; rather constants and variables are entered in the columns of the templates to construct an example related to the retrieval or update request.

QBE is based on domain calculus and has a two-dimensional syntax. The queries are written in the horizontal and vertical dimensions of a table. Queries are formed by entering an example of a possible answer in a skeleton (empty) table, as shown in fig. 3.3. This example contains variables as in domain calculus and specifies the conditions that have to be satisfied by the response. Conditions specified on a single row of the table are considered to be conjunctive i.e., "anded". Conditions entered on separate rows are dis-junctive (i.e., "ored").

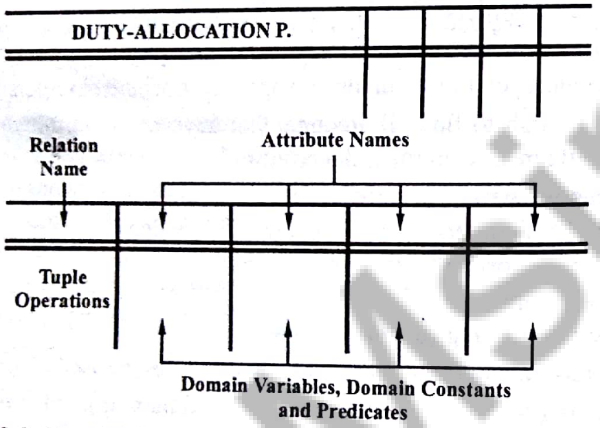
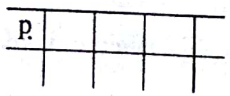


Fig. 3.3 QBE Skeleton Table for Writing QBE Example Query

The skeleton table does not have column headings. The first column is used for the relation name.

To get a list of relations, we enter p. for the PRINT command in the first column of the column heading –



To get the attribute names for a given relation we enter the relation name followed by a p.

Q.26. Differentiate between SQL and QBE. ***(R.G.P.V., June 2015)***

Ans. Refer to Q.2 and Q.25.

Q.27. How can the modification of database be done in QBE ?

Ans. Similar to SQL, modification of database can be done in QBE also. Modification can be done by any of the following –

(i) Insertion (ii) Deletion (iii) Updation.

If a row in a query has the operator I or D in the first column, then, when implementing the query, we do not create a tuple variable for this row. Rather, when a match for all the variables is found, we insert (I.) or delete (D.) into or from the relation in whose skeleton, one of these commands is found.

Variable in the row or rows to be inserted, deleted, or updated, take their values from the appropriate components of the tuple variables.

(i) Insertion – To insert the data into a relation we either specify a tuple to be inserted or write a query, the result of which is a set of tuples to be inserted. The insertion is done by placing the operator I. in the query expression. Obviously, the attribute value for inserting tuples must be members of the attributes domain.

The simplest insertion is as follows, in which we insert a tuple with values (1009, Peel, London, .30) in salespeople relation i.e, a salesperson Peel may have more than one snum, based on some criteria it also have different commissions (comm).

salespeople	snum	sname	city	comm
I.	1009	Peel	London	.30

"If we forget the city of Peel". Then we write,

salespeople	snum	sname	city	comm
I.	1009	Peel	_city	.30
		Peel	_city	

If we want to insert values in particular columns only, we do in following manner –

salespeople	snum	sname	city	comm
I.	1008	Ayush	Australia	

(ii) Deletion – The deletion of tuples from a relation is expressed in much the same way as a query. The major difference is the use of D. in place of P.

QBE provides us with better facility than SQL, as in QBE (unlike SQL), we can delete whole tuples as well as values in selected columns. When we

delete information in only some of the columns, null values, specified by –, a inserted. For example, delete salespeople where sname is Serres

salespeople	snum	sname	city	comm
D.		Serres		

If the value from particular column is to be deleted, for example, f deleting city of supplier Serres, we write

salespeople	snum	sname	city	comm
		Serres	D.	

Thus, as in our example, relation salespeople have the tuples (1002, Serre San Jose, .13) after deletion, it will have tuple (1002, Serres, –, .13). Th deletion can also be done from more than one relation. For example, if w wish to delete all the orders and customers with order of amount betwee 3000.00 and 5000.00. Then we write

customer	cnum	cname	city	rating	snum
D.	_y				

orders	onum	amt	odate	cnum	snum
D.		_x		_y	

Conditions
_x = (≥ 3000 and ≤ 5000)

(iii) Updates – QBE also allows the updation to be done. We can als update one value in a tuple without changing all values in the tuple. For thi purpose, we use U. operator. QBE does not allow users to update the primary key fields. For example – Suppose, we want to update rating of custome name (cname) 'Hoffman' to 150. We can write

customer	cnum	cname	city	rating	snum
		Hoffman		U.150	

The blank field of attribute cnum, city, and snum implies that no updating of that value is required. If we wish to update the rating of Hoffman using the previous value, we can request such request using two rows, one of them specifies old tuples that need to be updated and the other indicating the new updated tuples to be inserted in the database.

For example – If we want to increase all the ratings by 4 percent. We can write,

customer	cnum	cname	city	rating	snum
U.				_x * 1.04 _x	

The query retrieves one tuple at a time from the customer relation, determine the rating x and update that balance to x *1.04.

Q.28. How the aggregation operations can be applied in QBE ? Explain and give example.

Ans. QBE has the usual five aggregate operators, denoted by SUM, AVG, MAX, MIN and CNT (count). There are two other operators, ALL. and UNQ. (unique) that often are used in conjunction with aggregate operators. ALL., applied to a domain variable produces the multiset of values that the variable takes on as we run through all the tuples in the relevant relation. A multiset is a set with repetitions allowed, so the ALL. operator effectively leaves duplicates in, while most other QBE operations eliminate duplicates.

(i) Sum : SUM –

For example, if we wish to find the total amount of orders made by customer whose cnum is 2008. For this we write

orders	onum	amt	odate	cnum	snum
		P.SUM.ALL.		2008	

(ii) Average : AVG – This returns the average of numbers submitted to it as input. For example, if we wish to find the average of amounts of all the orders in order table.Then we write

orders	onum	amt	odate	cnum	snum
		P.AVG.ALL.			

QBE also offers the ability to compute functions on groups of tuples using the G operator which is analogous to SQL's group by construct. For example, if we wish to find average of amount separately for each snum. Then we write

orders	onum	amt	odate	cnum	snum
		P.AVG.ALL.			P.G.

Here, if we wish to represent all snum in ascending order. Then we write,

orders	onum	amt	odate	cnum	snum
		P.AVG.ALL.			P.AO.G.

Some conditions can also be applied here, as if we wish to find the average amount balance of only those snum whose average amount balance is more than 5000, then we write,

orders	onum	amt	odate	cnum	snum
		P.AVG.ALL._x			P.AO.G.

Conditions
AVG.ALL._x > 5000

(iii) Count : CNT – This operator counts the total number of tuples. For example, if we wish to find the total number of snum in order table. Then we should write,

orders	onum	amt	odate	cnum	snum
					P.CNT.ALL._x

This query would answer 10. The operator UNQ. converts a multiset into a set, by eliminating duplicates. For example, if we wish to find total number of suppliers in orders table. We can write

orders	onum	amt	odate	cnum	snum
					P.CNT.UNQ.ALL._x

In this way, before counting the set of suppliers produced by the expression ALL._x, the operator UNQ. removes duplicates. So, the above query would result 5.

Q.29. Explain the retrieval operations in QBE.

Ans. In QBE, retrieval queries are specified by filling in one or more rows in the templates of the tables. For a single relation query, we enter either constants or ***example elements*** in the columns of the template of that relation. An example element stands for a domain variable and is specified as an example value preceded by the underscore character (_). Additionally, a p. prefix (called the p dot operator) is entered in certain columns to indicate that we want to print values in those columns for our result. The constants specify values that must be exactly matched in those columns.

Queries on One Relation – To find all customer names who live in city London can be find as –

customer	cname	city	rating	snum
	P._x	London		

The query causes the system to look for tuples in customer that have city as London. For each such tuple, the value of cname attribute is assigned to the variable x. The value of variable x is printed (or displayed), because the command P appears in cname column next to the variable x. This is very similar to domain relational calculus query.

$$\{< x > | \exists n, C, R, S (< n, x, C, R, S > \in \text{customer} \wedge C = \text{"London"})\}$$

QBE (unlike SQL) performs duplicate elimination automatically. If we also, want to see all duplicate one's, then we insert the command all after the P.command.

customer	cnum	cname	city	rating	snum
		P.ALL.	London		

If we want to display the entire customer relation, then we can create a single row consisting of P in every field, as each field is desired in output. Another way is to put P in customer field i.e., a column headed by relation name.

customer	cnum	cname	city	rating	snum
P.					

Like SQL, the arithmetic comparisons are also possible in QBE, for example, if we want to find onum of all orders with an amount more than 900.

orders	onum	amt	odate	cnum	snum
	P.	> 900			

QBE supports many arithmetic operations such as, =, <, ≤, >, ≥ and ¬. If we want to find all the salespeople names who do not live in city London. Then query should be written as –

salespeople	snum	sname	city	comm
		P.	¬London	

Another example is, if we want to find all salespeople name who live in same city as Serres.

salespeople	snum	sname	city	comm
		P._x	_y	
		Serres	_y	

All of above queries are written on a single relation, but QBE also supports queries on more than one relation. For example, if we wish to find customer names who live in city in which salesman Peel lives.

salespeople	snum	sname	city	comm
		Peel	_x	

customer	cnum	cname	city	rating	snum
		P._y	_x		

According to above, result will have name of two customers i.e., 'Hoffman' and 'Clemens' as these two live in city 'London' in which salesperson 'Peel' resides. To evaluate this query, the system finds tuples in salespeople with salesperson name 'Peel' as the value for sname attribute. Then find the city of 'Peel'. Now, system finds all tuples in customers with city 'London'. The values of customer name i.e., cname attribute are displayed.

Condition Box – QBE allows conditions to be posted in queries. For this purpose we use a condition box. For example, if we want to find out only order numbers i.e., onum whose amount is between 700 and 5000, we can write

orders	onum	amt	odate	cnum	snum
	P.	_x			

Conditions

_x ≥ 700
_x ≤ 5000

QBE also allows logical expressions to appear in a condition box. The logical operators are the words ***and*** and ***or***, or the symbol "&" and "|" can also be used. For example, if we want to find the onum whose amount is between 700 and 5000, but not exactly 1900.00. We can write –

orders	onum	amt	odata	cnum	snum
	P.	_x			

Conditions

_x = (≥ 700 and ≤ 1500 and ¬1900)

For not exactly operation we use '¬' operator, which shows that, amount should not be equal to 1900. Another example, if we want to find all customers who live either in London or Rome, then we write –

customer	cnum	cname	city	rating	snum
		P.	_x		

Conditions

_x = (London or Rome)

Ordering of Tuples – Like SQL, QBE also allows to order (or sort) the tuples according to our requirement i.e., either in ascending order or in descending order. We get this control by inserting either the command AO (ascending order) or the command DO (descending order) in the appropriate column. For example, if we want to find name of all customers in customer table, but in descending order. Then we write,

customer	cnum	cname	city	rating	snum
		P.DO.			

The data can be sorted in multiple columns. For this, along with writing command, AO or DO, we have to specify the order in which sorting should be carried out by including an integer surrounded by parentheses. For example, if we want to find all customers of salesperson having snum = 1001 in ascending order and their rating sorted in descending order.

We can write

customer	cnum	cname	city	rating	snum
		P.AO(1).	P.	P.DO(2).	1001

The command P.AO(1). specifies that the customer names should be sorted first, the command P.DO(2). specifies that the rating for each customer should then be sorted.

NUMERICAL PROBLEMS

Prob.1. Consider the following relations with keys underlined –

Street (name, location, city)
House (number, street_name)
Lives (name, house_number)

(i) Define the above relations as tables in SQL making real world assumptions about the type of fields. Define the primary keys and foreign key.

(ii) For the above relations answer the following queries is SQL –

(a) Get the names of persons who live in the street named 'Mahatma Gandhi'.

(b) Get the house numbers street wise.

(c) Get the numbers of houses which are not occupied.

(R.G.P.V., June 2008)

Or

Consider the following relations with keys underlined –

Street (name, location, city)
House (number, street_name)
Lives (name, house_number)

wsinventry

For the above relations answer the following queries in SQL.

(i) Get the house numbers street wise.

(ii) Get the numbers of houses which are not occupied.

(R.G.P.V., June 2016)

Sol. (i) Tables in SQL

```
create table Street
    (name varchar (25) not null,
        location varchar (30),
        city varchar (15),
        primary key (name));
create table House
        (number int not null,
        street_name varchar (25) not null,
        primary key (number),
        foreign key (street_name) references Street (name));
        create table Lives
        (name varchar (25) not null,
        house_number int not null,
        primary key (name),
        foreign key (house_number) references House (number));
```

(ii) (a) The SQL statement is as follows –

```
select name from House, Lives where
House.number = Lives.house_number AND
street_name = 'Mahatma Gandhi';
```

(b) The SQL statement is as follows –

```
select number, street_name from House order by street_name;
```

(c) The SQL statement is as follows –

```
select number from House where number not in
(select house_number from Lives);
```

Prob.2. Consider the following relational database –
Employee (Person-name, street, city)
Works (Person-name, company-name, salary)
Company (Company-name, city)
Manages (Person-name, manager name)
Give SQL DDL definitions of this database.
Identify referential integrity constraints that should hold and include them in DDL definitions and solve the following queries in SQL –

(i) Find the names of all employees who work in "First BC".

(ii) Find all employees who live in the city where company for which they work is located.

(iii) Give all managers 20% raise.

(iv) Delete all types in 'works' relation for employees of "First BC".

(v) Find the company that has smallest payroll.

(R.G.P.V., Dec. 2009)

Sol. The SQL DDL statements are as follows –

```
create table Employee
(Person_name        varchar (20)        Not Null,
street              varchar (40),
city                varchar (10),
primary key (Person_name));
create table Company
(company_name       varchar (20).       Not Null,
city                varchar (10),
primary key (company_name));
create table Works
(Person_name        varchar(20)         Not Null,
company_name        varchar (20),
salary              integer (10),
foreign key (Person_name) references Employee,
foreign key (company_name) references Company,
check (salary >  0));
create table Manages
(Person_name        varchar (20)        Not Null,
manager_name        varchar (20),
foreign key (person_name) references employee);
```

The SQL queries are as follows –

(i) Select Person_name from Works
where company_name = "First BC ",

(ii) Select Person_name from Employee, Company
where Employee.city = Company.city;

(iii) Update Works set salary = salary*1.2
where Person_name in (select Person_name from Works, Manages where Works.Person_name = Manages.manager_name);

(iv) delete from Works where company_name = "First BC";

(v) Select company_name, min (salary) from Works group;

Prob.3. Consider the relational database given below –
lives (person_name, street, city)
works (person_name, company_name, salary)
located in (company_name, city)
manager (person_name, manager_name)
Give an expression in SQL for each of the queries below –

(i) Find the company with the smallest payroll

(ii) Find the company with the most employees

(iii) Give all managers a 10% raise.

(R.G.P.V., June 2015)

Visinvenier

Sol. The SQL queries are as follows –

(i) Refer to Prob.2 (v).

(ii) Select company_name from works groupby company_name having count (distinct person_name) > = all (select count (distinct person_name) from works groupby company_name)

(iii) Update works set salary = salary *1.1 where person_name in (select person_name from works, manager where works.person_name = manager.manager_name).

Prob.4. Given the relation schema –

S_emp (Id, Last_name, First_name, Userid, Start-date, Designation, Dept_id, Salary)

Give queries expressed in SQL –

(i) Display the last name, first name and department Id for all employees who do not work in department 44.

(ii) Display the last name, user id and start date for all employees hired on March 8, 1990.

(iii) Display the last name, first name and salary of all employees whose salary is less than $ 1,200.

(iv) Display the last name, first name and salary of all the employees who are managers. Order the result in ascending order by salary.

(v) Increase the salary of all employees of department 41 by 1000.

(R.G.P.V., Dec. 2011)

Sol. (i) The SQL expression is as follows –

```
select Last_name, First_name, Dept_id
from S_emp
where Dept_id! = 44;
```

(ii) The SQL expression is as follows –

```
select Last_name, Userid, Start-date
from S_emp
where Start-date = '8-MAR-1990';
```

(iii) The SQL expression is as follows –

```
select Last_name, First_name, Salary
from S_emp
where Salary < 1200;
```

(iv) The SQL expression is as follows –

```
select Last_name, First_name, Salary
from S_emp
where Designation = 'managers'
order by Salary asc;
```

(v) The SQL expression is as follows –

```
update S_emp
set Salary = Salary + 1000
where Dept_id = 41;
```


Prob.5. Consider the following relational database –

Employee (employee_name, street, city)

Works (employee_name, company_name, salary)

Company (company_name, city)

Manages (employee_name, manager_name)

Write SQL for the following queries –

(i) Find the names of all employees in this database who live in the same city as the company for which they work.

(ii) Find the names of all employees who live in the same city and on the same street as do their manager.

(iii) Find the names of all employees in this database who do not work for the company 'FBC'.

(R.G.P.V., Dec. 2008)

Sol. (i) The SQL expression is as follows –

```
select employee_name from Employee, Works, Company
where Employee.employee_name = Works.employee_name
and Works.company_name = Company.company_name
and Employee.city = Company.city;
```

(ii) The SQL expression is

```
select emp_name from Employee
where Employee.street AND Employee.city IN
(select street, city from Employee, Manages
where emp_name = manager_name)
Its corresponding QUEL statement is
range of e is Employee
range of m is Manages
retrieve (e.emp_name) where e.street AND e.city IN
(retrieve (e.street, e.city) where e.emp_name = m.manager_name)
```

(iii) The SQL expression is as follows –

```
select employee_name from Works where
company_name ≠ 'FBC';
```

Prob.6. Consider the relational database –

employee (empname, street, city)

works (empname, companyname, salary)

company (companyname, city)

manages (empname, managername)

Give an expression in SQL for the following queries –

(i) Find the name of all employees who work for First Bank Corporation.

(ii) Find the names, street addresses and cities of residence of all employees who work for First Bank Corporation and earn more than 2,00,000 per month.

(iii) Find the names of all employees who live in the city as the companies for which they work.

(R.G.P.V., June 2010)

Msinvertor

Sol. (i) select empname from works
where companyname = 'First Bank Corporation';

(ii) select empname, street, city, from employee, works
where employee.empname = works.empname
and companyname = 'First Bank Corporation'
and salary > 200000;

(iii) Refer to Prob.5 (i).

Prob.7. Solve the following query with "SQL" and "Relational Algebra". Consider the given database –

S (Sno, Sname, Status, City)
P (Pno, Pname, color, Weight)
SP (Sno, Pno, Qty)

(i) Get supplier names for suppliers who supply part P2.

(ii) Get supplier numbers for suppliers who supply at least one red part.

(iii) Get supplier number for supplier who supply at least all those parts supplied by supplier S2.

(iv) Get supplier names for suppliers who do not supply part P2.

(v) Get supplier number who supplies maximum quantity.

(R.G.P.V., June 2009)

Sol. (i) SQL Statement –

select Sname from S, SP where Sno in (select Sno from SP where Pno = 'P2');

Relational statement –

((SP JOIN S) WHERE Pno = Pno ('P2')) {Sname}

(ii) SQL Statement –

select Sno from SP where Pno is (select Pno from P where color = 'red');

Relational statement –

(((P where color = color ('red')) JOIN SP) {Sno} JOINS) {SName}

(iii) SQL Statement –

select Sno from SP where Sno <> 'S2' and P no is (select Pno from SP where Sno = 'S2');

Relational Statement –

S{S#} DIVIDEBY (SP where Sno = Sno('S2')) {Pno} per SP{Sno, Pno}

(iv) SQL Statement –

select distinct S.sname from S where S.Sno not in (select SP.Sno from SP where SP.Pno = 'P2')

Relational Statement –

((S{Sno} minus (SP where Pno=Pno ('P2')) {Sno}) Join S) {Sname}

(v) SQL Statement –

select Sno, max(x) from (select Sno, sum(SP.Qty) As x from SP group by SP.Sno);

Relational Statement – Sno $Gsum_{Qty}$, max_{Qty} (SP)

Prob.8. Consider the following database (supplier part database) with primary key underlined –

S(SNO, SName, city, status)
P(PNO, PName, city, weight, color)
SP(SNO, PNO, Qty) relational algebra

Give expression in QBE and SQL for the following question –

(i) Get SNames for suppliers who do not supply part 'P2'.

(ii) Double the status of 'LONDON' suppliers.

(iii) Delete all 'PARIS' suppliers and corresponding shipments.

(iv) Get SNames for suppliers who supply at least one 'red' part.

(v) Get all pairs of suppliers who are located in same city.

(vi) Get SNames for suppliers who supply at least one part supplied by supplier 'S2'.

(R.G.P.V., June 2007)

Sol. (i) The SQL expression is as follows –

```
Select distinct S.SName
from S
where NOT EXISTS
    (select *from SP
    where S.P.SNO = S.SNO
    AND SP.PNO = 'P2');
```

Its corresponding QBE expression is as follows –

P	PNO	PName	city	weight	color
	_x	P2			

SP	SNO	PNO	Qty
	_y	_x	

S	SNO	SName	city	status
	_y	_z		

Result	SName
P	_z

MSInvest

(ii) The SQL expression is as follows –

```
update S
set status = 2*status
where city = 'LONDON';
```

Its corresponding QBE expression is as follows –

S	SNO	SName	city	status
			LONDON	U._x*2

(iii) The SQL expression is as follows –

```
Delete from SP
where city = 'PARIS' AND SP.SNO IN (select SNO from S where
S.SNO = SP.SNO);
```

Its corresponding QBE expression is as follows –

S	SNO	SName	city	status
D.	_y		_x	

SP	SNO	PNO	Qty
P.	_y		

Conditions
_x = 'PARIS'

(iv) The SQL expression is as follows –

```
select distinct S.SName
From S
where S.SNO in
(select SP.SNO
from SP
where SP.PNO in
(select P.PNO from P where P.color = 'red'));
```

Its corresponding QBE expression is as follows –

P	PNO	PName	city	weight	color
	_y				_x

SP	SNO	PNO	Qty
	_z	_y	

S	SNO	SName	city	status
P.	_x	_w		

Conditions
_x = 'red'

(v) The SQL expression is as follows –

```
select A.SNO as SA, B.S# as SB
from S as A, S as B
where A.city = B.city
AND A.SNO < B.SNO;
```

Its corresponding QBE expression is as follows –

S	S#	city
	_SX	_CZ
	_SY	_CZ

P.	_SX	_SY

(vi) The SQL expression is as follows –

```
select SName from S, SP
where S.SNO = SP.SNO and PNO in (select PNO from SP where
SNO = 'S2')
```

Its corresponding QBE expression is as follows –

S	S#	SName
	_SX	P._NX

SP	S#	P#
	_SX	_PX

SP	S#	P#
	S2	_PX

Prob.9. Solve the following on the given database. Write the SQL and Relational Algebra of given queries –

S (Sno, Sname, City, Status)

P (Pno, Pname, Color, Weight)

SP (Sno, Pno, Qty)

(i) Get supplier name who supplies at least one red part.

(ii) Get supplier number who supplies maximum quantity.

(iii) Get supplier number who supplies quantity greater than average quantity.

(iv) Increase the quantity of part P_1 by 10%.

(v) Change the color of Red parts supplied by Ajay to Green.

(R.G.P.V., Dec. 2010, June 2012)

Sol. The SQL of given queries is as follows –

(i) Refer to Prob.8 (iv).

(ii) Refer to Prob.7 (v).

(iii) Select Sno from SP
where
Qty > Select avg (Qty) from SP group by Sno;

(iv) Update SP
Set Qty = Qty + Qty * 0.10
where Pno = P'_1;

MSinverter

(v) Update P
Set color = 'Green'
where color in (Select color
from S, P, SP
and S.Sno = SP.Sno
and SP.Pno = P.Pno
and P.color = 'Red'
and S.Sname = 'Ajay')

The relational algebra statements are as follows –

(i) (((P where color = color ('Red'))
JOIN SP) {Sno} JOIN S) {SName}

(ii) $_{Sno}\mathcal{G}_{max\ (Qty)}$ (SP)

(iii) $_{Sno}\mathcal{G}_{Qty\ >\ avg\ (Qty)}$ (SP)

(iv) SP ← $\Pi_{Sno,\ Pno,\ Qty\ *\ 1.1}$ ($\sigma_{Pno\ =\ 'P1'}$ (SP))

(v) P ← $\Pi_{Pno,\ Pname,\ color\ =\ 'Green',\ Weight}$ (σSname='Ajay'(σ(σcolor='Red' (P)) JOIN SP) JOIN S)

Prob.10. Consider the following database with primary keys underlined–

Employee (ENO, DOB, ADDRESS, SEX, SALARY, NAME, DN0 SUPER END).

Department (DNO, DName, MGRENO, MGR start date)

Dept-location (DNO, Dlocation)

Project(PNAME, PNO, Plocation, DNO)

Works-ON(ENO, PNO, hours)

Dependent (ENO, dep-name, sex, bdate, relationship)

For each of the following queries give expressions in SQL and relational algebra –

(i) Retrieve the names of all employees in department 5.

(ii) List the names of all employees who have no dependents.

(iii) Retrieve the names of all employees who do not work on any project.

(iv) Retrieve the average salary of all female employee.

(v) Find names of all employees who work on at least one project located in Hyderabad.

(vi) Write SQL, DDL statements for the above database.

(R.G.P.V., Nov./Dec. 2007)

Sol. (i) The SQL expression is as follows –
select NAME from Employee where DNO = 5
Its corresponding relational algebra expression is as follows –
$\Pi_{NAME}(\sigma_{DNO=\ 5}(Employee))$

(ii) The SQL expression is as follows –
select NAME from Employee
where not exists
(select *from Dependent where Employee.ENO = Dependent.ENO)

Its corresponding relational algebra expression is as follows –
ALL_EMPS ← Π_{ENO} (Employee)
EMP_WITH_DEPS(ENO) ← Π_{ENO} (Dependent)
EMP_WITHOUT_DEPS ← (ALL_EMPS_EMPS_WITH_DEPS)
RESULT ← Π_{NAME} (EMP_WITHOUT_DEPS*Employee)

(iii) The SQL expression is as follows –
select NAME from Employee
where not exists(select * from Works-ON
where Employee.ENO = Works-ON.ENO)

Its corresponding relational algebra expression is as follows –
$\Pi_{ENO,NAME}$(Employee)_Π_{ENO} (Works_ON)

(iv) The SQL expression is as follows –
select avg(SALARY)from Employee where SEX = 'female'

Its corresponding relational algebra expression is as follows –
$\mathcal{G}_{avg(SALARY)}$ ($\sigma_{SEX\ =\ 'female'}$ (Employee))

(v) The SQL expression is as follows –
select NAME from Employee, Project where ENO in
(select * from Project, Works-ON where
Project.PNO = Works-ON.PNO
AND Plocation = 'Hyderabad')

Its corresponding relational algebra expression is as follows –
Π_{NAME} ($\sigma_{Project\cdot PNO\ =\ Works_ON.PNO}$
($\sigma_{Plocation\ =\ "Hyderabad"}$ (Project X works_ON)))

(vi) The SQL, DDL statements are as follows –

```
Create table Employee
 (ENO          INT            NOT NULL,
 DOB           DATE,
 ADDRESS       VARCHAR(30),
 SEX           VARCHAR(6),
 SALARY        DECIMAL(10, 2)
 NAME          VARCHAR(30),
 DNO           INT            NOT NULL,
 SUPEREND      INT            NOT NULL,
PRIMARY KEY (ENO));
Create table Department
 (DNO      INT          NOT  NULL,
 DNAME     VARCHAR(15)  NOT  NULL,
 MGRENO    CHAR(9)      NOT  NULL,
 MGR start date  DATE,
```

Mrinvent.in

```
Primary Key (DNO),
Unique (DName),
FOREIGN KEY(MGRENO) References Employee (ENO));
Create table Dept_location
    (DNO        INT              NOT  NULL,
     Dlocation  VARCHAR(15)      NOT  NULL,
     Primary key (DNO));
Create table Project
    (PNAME      VARCHAR(15)      NOT NULL,
     PNO        INT              NOT NULL,
     Plocation  VARCHAR(15),
     DNO        INT              NOT NULL,
     Primary key (PNO),
     UNIQUE (PNAME)
     FOREIGN KEY (DNO) REFERENCES Department (DNO));
Create table Works_ON
    (ENO        CHAR(9)          NOT  NULL
     PNO        INT              NOT  NULL
     HOURS      DECIMAL (3, 1)   NOT  NULL,
Primary key (ENO, PNO),
FOREIGN key (ENO) References Employee (ENO),
FOREIGN key (PNO) References Project (PNO));
Create table Dependent
    (ENO        CHAR(9)          NOT NULL,
     dep_name   VARCHAR(15)      NOT NULL,
     sex        VARCHAR(6),
     bdate      DATE,
    relationship VARCHAR(8),
PRIMARY KEY (ENO, dep_name),
FOREIGN KEY (ENO) References Employee (ENO));
```

Prob.11. Consider the following schema for a company database

Employee (Name, SSN, Address, Sex, Salary, Dno)

Department (Dname, Dnumber, MGRSSN, MGRSTART Date)

Dept-Locations (Dnumber, Dlocations)

Project (Pname, Pnumber, Plocations, Dnum)

Works-On (ESSN, PNo, Hours)

Dependent (ESSN, Dependent-name, Sex, Bdate, Relationship)

Give the queries in SQL

(i) Retrieve the names and address of employees who work for "Research" Department.

(ii) List all the project names on which employee "Smith" is working.

(iii) Retrieve all employees who either work in department 4 and make over 25000 per year or work in department 5 and make over 30,000.

(iv) Retrieve the SSN of all employees who either work in department 5 or directly supervise an employee who works in department number.

(R.G.P.V., June 2017)

Sol. (i) Select Name, Address From Employee, Department
where Employee.Dno = Department.Dnumber
and Dname = 'Research';

(ii) Select Pname from Project, Employee
where Employee.Dno = Project.Dnum
and Name = 'Smith';

(iii) Select Name from Employee
where (Dno = '4' and 12*Salary > 25000) or
(Dno = '5' and 12*Salary > 30000);

(iv) Select SSN from Employee, Department
where Employee.Dno = '5' or
Employee.SSN = Department.MGRSSN;

Prob.12. Employee (SSN, name, age, dno)

Salary (SSN, salary)

Works on (Projects#, SSN)

Project (Projects#, project_name, location)

For each of the following queries give an expression in SQL –

(i) Display the names of projects at "Delhi".

(ii) Find the project_name of employee whose salary is greater than 10000.

(iii) Retrieve the name and SSN of employees working on Project# A100.

(R.G.P.V., Dec. 2011)

Sol. (i) The SQL query is as follows –
Select project_name from Project
where location = "Delhi"

(ii) Select project_name from Project where Projects# in (select Projects# from Works_on, Salary where Works_on.SSN = Salary.SSN having Salary > 10000);

(iii) Select name, SSN from Employee,
where SSN in (select SSN from Works_on where Projects# = A 100);

MSInventer

Prob.13. For the given database, write SQL queries –

employee (eid, employee_name, street, city)
works (eid, cid, salary)
company (cid, company_name, city)
manager (eid, manager_name)

(i) Find the names, street, and city of all employees who work for 'ABC' and earn more than Rs. 20,000.

(ii) Find the names of all employees having "i" as the second letter in their names.

(iii) Display the annual salary of all employees.

(R.G.P.V., June 2014)

Sol. (i) Select employee_name, street, city from employee where eid in (select eid from works, company where company_name = "ABC" and salary > 20000)

(ii) Select employee_name from employee
where employee_name like "0_i%";

(iii) Select eid, 12 * salary from works;

Prob.14. Consider the following relations –

Student (snum : integer, sname : string, major string, level : string, age : integer)

Class (name : string, meetsat : string, room : string, fid : integer)

Enrolled (snum : integer, cname : string)

Faculty (fid : integer, fname : string, deptid : integer)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL –

(i) Find the names of all classes that either meet in room R128 or have five or more students enrolled.

(ii) Find the names of all students who are enrolled in two classes that meet at the same time.

(R.G.P.V., Dec. 2015)

Sol. (i)
```
SELECT C.name
FROM Class C
WHERE C.room = 'R128'
OR C.name IN (SELECT E.name
FROM Enrolled E
GROUP BY E.Cname
HAVING COUNT (*) > = 5)
```


(ii)
```
SELECT DISTINCT S.sname
FROM Student S
WHERE S.sname IN (SELECT E1.snum
FROM Enrolled E1, Enrolled E2, Class C1, Class C2
WHERE E1.snum = E2.snum AND E1.cname < > E2.cname
AND E1.cname = C1.name
AND E2.cname = C2.name AND C1.meetsat = C2.meetsat
```

Prob.15. Write SQL statement for the following –

Student (Enrno, Name, CourseID, EmailID, Cell No.)
Course (CourseID, Course-nm, duration)

(i) Add a column city in student table.

(ii) Find out list of students who have enrolled in "computer" course.

(iii) List name of all course with their duration.

(iv) List name of all students starting with "a".

(v) List EmailID and cell no. of all mechanical engineering students.

(R.G.P.V., May 2018)

Sol. (i) ALTER TABLE Student ADD city VARCHAR (12);

(ii) SELECT Name from Student
WHERE CourseID.Student.IN (SELECT CourseID from Course WHERE Course-nm = "computer");

(iii) SELECT c.Course-nm, c.duration from Course As c;

(iv) SELECT s.Name from Student As s WHERE Name like 'a %';

(v) SELECT EmailID, Cell No from Student where CourseID, Student IN (SELECT CourseID from Course WHERE Course-nm = "mechanical engineering") ;

QUERY PROCESSING & QUERY OPTIMIZATION – OVERVIEW, MEASURES OF QUERY COST, SELECTION OPERATION, SORTING, JOIN, EVALUATION OF EXPRESSIONS, TRANSFORMATION OF RELATIONAL EXPRESSIONS, ESTIMATING STATISTICS OF EXPRESSION RESULTS, EVALUATION PLANS

Q.30. What do you understand by query processing ?

Ans. Query processing refers to the activities involved in extracting data from a database. These activities are translation of queries in high level database language into expressions, a query optimizing transformation and evaluation of queries.

Mishver

Q.31. Write the steps involved in query processing.

Ans. Fig. 3.4 shows the steps involved in processing a query. These ste
are –

(i) Parsing and translation (ii) Optimization (iii) Evaluation.

The system must translate the query into a usable form before que
processing. A language like SQL is appropriate for human use, but is not goo
to be the system's internal representation of a query. A more useful interna
representation depends on the extended relational algebra.

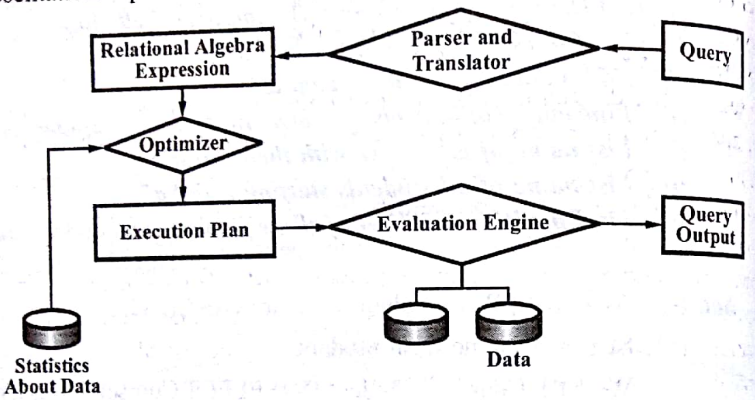


Fig. 3.4 Steps in Query Processing

Thus, to translate a given query into its internal form is the first actio
the system takes in query processing. This translation process is identical t
the work done by the parser of a compiler. The parser checks the syntax o
the user's query in generating the internal form of the query, verifies that th
relation names present in the query are names of the relations in the databas
and so on. A parse-tree representation of the query is constructed by th
system, then the system translates into a relational-algebra expression. Th
translation phase also replaces all uses of the view by the relational-algebr
expression defining the view, if the query was expressed in terms of a vie

We require not only to provide the relational algebra expression, but als
to annotate it with instructions specifying how to evaluate each operation
specify fully how to evaluate a query. Annotations say the algorithm to be use
for a particular operation, or the particular index or indices to use. An ***evaluatio***
primitive is a relational algebra operation with instructions on how to evalua
it. A ***query-execution plan*** or query-evaluation plan is a sequence of primiti
operations that are used to evaluate a query. The query is evaluated with th
plan, and the result of the query is output, when the query plans is selecte
The sequence of steps is representative and is already described for processin
a query. These steps are not followed exactly by all databases.

A query optimizer must know the cost of each operation to optimize a query. However, the exact cost is difficult to compute, because it is based on many parameters like actual memory available to the operation, it is possible to find a rough estimate of execution cost for each operation.

Q.32. Discuss the different types of parameters that are used in cost functions. Where is this information kept ?

Ans. To estimate the costs of various execution strategies, we must keep track of any information that is required for the cost functions. This information may be stored in the DBMS catalog, where it is accessed by the query optimizer. First, we must know the size of each file. For a file, whose records are all of the same type, the number of records (tuples) (r), the (average) record size (R), and the number of blocks (b) are required. The blocking factor (bfr) for the file may also be needed. We must also keep track of the primary access method and the primary access attributes for each file. The file records may be unordered, ordered by an attribute with or without a primary or clustering index, or hashed on a key attribute. Information is kept on all secondary indexes and indexing attributes. The number of levels (x) of each multilevel index is required for cost functions that estimate the number of block accesses that occur during query execution. In some cost functions the number of first-level index blocks (b_{11}) is needed.

Another important parameter is the number of distinct values (d) of an attribute and its selectivity (sl), which is the fraction of records satisfying an equality condition on the attribute. This permits estimation of the selection cardinality ($s = sl*r$) of an attribute, which is the average number of records satisfying an equality selection condition on that attribute. For a key attribute, $d = r$, $sl = 1/r$ and $s = 1$. For a nonkey attribute, by making an assumption that the d distinct values are uniformly distributed among the records, we estimate $sl = (1/d)$ and so $s = (r/d)$.

Q.33. What is query optimization ? Discuss various steps of optimization.

Ans. A query has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as query optimization. One aspect of optimization takes place at the relational algebra level, where the system tries to find an expression that is equivalent to the given expression, but more efficient to execute. Second aspect is to choose a detailed strategy for processing the query, like selecting the algorithm to use for executing an operation, selecting the specific indices to use, and so on. A query expressed in a high-level query language such as SQL must first be scanned, parsed, and validated. The scanner identifies the language tokens, such as SQL keywords, attribute names, and relation names, in the text of the query, whereas the parser checks the query syntax to determine whether it is formulated according

to the syntax rules (i.e., rules of grammar) of the query language. The query must also be validated, by checking that all attribute and relation names are valid and semantically meaningful names in the schema of the particular database being queried. An internal representation of the query is then created as a tree data structure called a ***query tree.*** It is also possible to represent the query using a graph data structure called a ***query graph.*** The DBMS must then device an execution strategy for retrieving the result of the query from the database files.

Fig. 3.5 shows the various steps of optimization. The ***query optimizer*** module has the task of producing an execution plan, and the ***code generator*** generates the code to execute that plan. The ***runtime database processor*** has the task of running the query code, whether in compiled or interpreted mode, to produce the query result. If a runtime error results, an error message is generated by the runtime database processor.

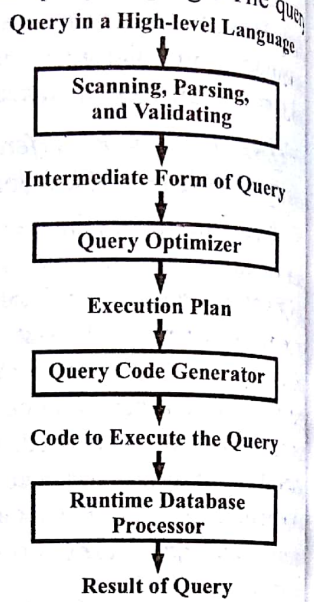


Fig. 3.5 Steps of Optimization

Q.34. Give a brief note on query processing and query optimization.
(R.G.P.V., May 201

Ans. Refer to Q.30 and Q.33.

Q.35. What is meant by cost-based query optimization ?

Ans. A query optimizer should also estimate and compare the costs of executing a query using different execution strategies and should choose the strategy with the lowest cost estimate. For this approach to work, accurate cost estimates are required so that different strategies are compared fairly and realistically. In addition, we must limit the number of execution strategies to be considered, otherwise too much time will be spent making cost estimates for the many possible execution strategies. This approach is called as cost based query optimization.

Q.36. What is meant by the term heuristic optimization ?
Or
Explain in detail about heuristic approach of query optimization.
(R.G.P.V., Dec. 201

Ans. The heuristic optimization is a technique that apply heuristic rules to modify the internal representation of a query, which is usually in the form of

query tree or a query graph data structure, to improve its expected performance. The parser of a high-level query first generates an initial internal representation, which is then optimized according to heuristic rules. Following that, a query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.

Q.37. Discuss the main heuristics that are applied during query optimization.

Ans. The main heuristic is to apply first the operations that reduce the size of intermediate results. This includes performing as early as possible SELECT operations to reduce the number of tuples and PROJECT operations to reduce the number of attributes. This is done by moving SELECT and PROJECT operations as far down the tree as possible. In addition, the SELECT and JOIN operations that are most restrictive, i.e. result in relations with the fewest tuples or with the smallest absolute size, should be executed before other similar operations. This is done by reordering the leaf nodes of the tree among themselves while avoiding cartesian products, and adjusting the rest of the tree appropriately.

Q.38. How would you measures the query cost ?

Ans. A number of different resources, including CPU time to execute a query, disk access, and, in a distributed or parallel database system, the cost of communication measure the cost of query evaluation. Assuming no other activity is going on the computer, the response time for a query evaluation plan would account for all these costs, and is used as a good measure of the cost of the plan.

However, the most important cost in large database systems is disk accesses, because disk accesses are slower than in-memory operations. Furthermore, CPU speeds are improving much faster as compared to disk speeds. Therefore, it is possible that the time taken in disk activity will dominate the total time to run a query, Finally, estimating the CPU time is difficult than to estimate the disk-access cost. Hence, disk-access cost is considered a good measure of the cost of a query evaluation plan.

A measure of the actual cost is the number of block transfers. It is assumed that all transfers of blocks have the same cost, which simplifies disk-access cost computation. The variance due to seek time and rotational latency is neglected by this assumption. The sequential I/O is where the blocks read are contiguous on disk, and the random I/O is where the blocks are noncontiguous. For each disk I/O operation, an extra seek cost is paid. It is needed to distinguish between reads and writes of blocks, because it spent more time to write a block to disk compared to read a block from disk. Hence, a more exact measure estimates the number of blocks written, the number of blocks read, and the number of seek operations performed. Now, these numbers are added up

after multiplying them by the average transfer time for writing a block, the average transfer time for writing a bock, and the average seek time, respectively. When computing the cost of an operation, real-life query optimizers also consider CPU costs.

The cost of writing the final result of an operation back to disk is neglected by the cost estimates. These are considered separately when needed. The cost of all the algorithms, which are considered, is based on the size of the buffer in main memory. All data are read into the buffers, and the disk does not require to be accessed again in the best case. It is assumed that the buffer holds only a few blocks of data in the worst case, approximately one block per relation. In general the worst case is assumed when presenting cost estimates.

Q.39. What are the steps involved in query processing ? How would you estimate the cost of the query ? ***(R.G.P.V., June 2011)***

Ans. Refer to Q.31 and Q.38.

Q.40. Discuss the cost components for a cost function that is used to estimate query execution cost. Which cost components are used most often as the basis for cost functions ?

Ans. The cost of executing a query involves the following components-

(i) Access Cost to Secondary Storage – This is the cost of searching for reading, and writing data blocks that reside on secondary storage, mainly on disk. The cost of searching for records in a file depends on the type of access structures on that file such as ordering, hashing, and primary or secondary indexes. In addition, the factors such as whether the file blocks are allocated contiguously on the same disk cylinder or scattered on the disk affect the access cost.

(ii) Storage Cost – This is the cost of storing any intermediate files that are generated by an execution strategy for the query.

(iii) Memory Usage Cost – This is the cost pertaining to the number of memory buffers required during query execution.

(iv) Computation Cost – This is the cost of performing in memory operations on the data buffers during query execution. These operations are searching for and sorting records, merging records for a join, and performing computations on field values.

(v) Communication Cost – This is the cost of shipping the query and its results from the database site to the site or terminal where the query originated.

For large databases, the main emphasis is on minimizing the access cost to secondary storage. Simple cost functions ignore other factors and compare different query execution strategies in terms of the number of block transfers

between disk and main memory. For smaller databases, the emphasis is on minimizing computation cost. In distributed databases, communication cost must be minimized. It is difficult to include all the cost components in a cost function because of the difficulty of assigning suitable weights to the cost components. That is why some cost functions consider a single factor only – disk access.

Q.41. How can you perform selection operation using file scan ?

Or

Discuss file scan and its algorithms of selection operation.

Ans. File scans are search algorithms that locate and retrieve records that satisfy a selection condition. It is the lowest level operator to access data in query processing. A file scan, in relational systems, allows an entire relation to be read if it is stored in a single dedicated file. There are two scan algorithms for selection operation, which are as follows –

(i) Linear Search – In linear search, the system scans each file block and tests all records to check that they fulfill the selection condition. The system can terminate the scan if the key record is found without looking at the other records. The cost of linear search in terms of number of I/O operations is b_r, where b_r is the number of blocks in the file. The linear search algorithm can be applied to any file independent of the ordering of the file, or the availability of indices, or the nature of the selection operation. This algorithm is slower than other algorithms. The other algorithms are not applicable in all cases but when applicable they perform faster as compared to linear search.

(ii) Binary Search – A binary search is use if the file is ordered on an attribute and the selection condition is an equality comparison on the attribute. The blocks number that need to be scanned to search a block containing the key records is $[\log_2 (b_r)]$. If the selection is on a nonkey attribute, more than one block may contain required records, and the cost of reading the extra blocks has to be added to the cost estimate.

Q.42. What is sorting ?

Ans. Sorting of data plays a vital role in database systems. First, SQL queries can specify that the output be sorted. Second, several relational operations can be implemented efficiently if the input relations are first sorted. A relation can be sorted by making an index on the sort key and then using it to read the relation in sorted order. However, such a process orders the relation only logically through an index. Thus, the reading of tuples in the sorted order may lead to a disk access for each record which can be very expensive. That's why, it is required to order the records physically rather that logically. Sorting can be of two types – internal sorting and external sorting. Sorting of relation that fit entirely in main memory is called internal sorting. The most commonly used technique for internal sorting is quick sort. Sorting of relations

that do not fit in memory is called external sorting. The most commonly use technique for external sorting is the external sort merge algorithm.

Q.43. What is join operation ? How it can be performed on a relation

Ans. We use the term equi-join to refer to a join of the form r $\bowtie_{r.A = s.B}$ where A and B are attributes or sets of attributes of relations r and s respectively. Depending on the availability of indices and the form of physical storage for the relations, queries involving a natural join may be performed in various ways –

(i) A block nested loop join strategy may be advantageous if the join result is almost as large as the cartesian product of the two relations.

(ii) The indexed nested loop join can be used if indices are available

(iii) A merge join may be desirable if the relations are sorted. It may be advantageous to sort a relation prior to join computation.

(iv) The hash join algorithm divides the relations into several pieces in such a way that each piece of one of the relations fits in memory. The partitioning is carried out with a hash function on the join attributes, so that corresponding pairs of partitions can be joined independently.

Q.44. Describe evaluation of expressions by materialization approach

Ans. It is easy to evaluate an expression by looking at a pictorial representation of the expression in an operator tree. For example, the expression

$$\pi_{c_name}(\sigma_{Amt > 20000}(account) \bowtie customer)$$

is shown in fig. 3.6. We start from the lowest level operations in the expression in materialization. In our expression, there is only one such operation. The selection operation on account. Relations in the database are provided as input to the lowest level operations. These operations are executed by the algorithm and the results are store in temporary relations which are used to execute the operations at the next level up in the tree. In our expression, the inputs to the join are the temporary relation created by the selection on account and the customer relation. We will eventually evaluate the operation at the root of the tree by repeating the process and providing the final result of the expression. This type of evaluation is called materialized evaluation since the result of each intermediate operation are materialized (created) and then used for evaluation of the next level operations. A disadvantage of this approach is the need to construct the temporary relation which must be written to disk.

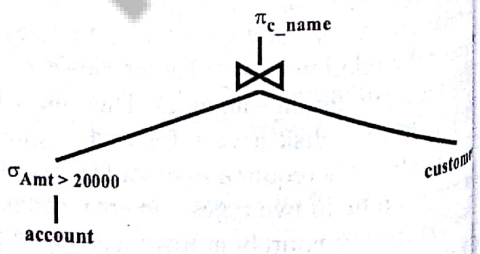


Fig. 3.6

Q.45. Discuss the different algorithms for each of the following relational algebra operations –
(i) Select (ii) Join (iii) Project.

Ans. (i) Select Operation – There are many options for executing a SELECT operation. Some depend on the file having specific access paths and may apply only to certain types of selection conditions. Some of the algorithms for implementing SELECT are discussed using the following example operations –

(op1) – $\sigma_{SSN = '12345'}$ (EMPLOYEE)
(op2) – $\sigma_{DNUMBER > 5}$ (DEPARTMENT)
(op3) – $\sigma_{DNO = 5}$ (EMPLOYEE)
(op4) – $\sigma_{DNO = 5 \text{ AND } SALARY > 30000 \text{ AND } SEX = 'F'}$ (EMPLOYEE)
(op5) – $\sigma_{ESSN = '12345' \text{ AND } PNO = 10}$ (WORKS_ON)

Search Methods for Simple Selection – The following search methods are examples of some of the search algorithms that can be used to implement a select operation –

(a) Linear Search (brute force) – Retrieve every record in the file, and test whether its attribute values satisfy the selection condition.

(b) Binary Search – If the selection condition involves an equality comparison on a key attribute on which the file is ordered, binary search can be used. An example is op1 if SSN is the ordering attribute for the EMPLOYEE file.

(c) Using a Primary Index (or hash key) – If the selection condition involves an equality comparison on a key attribute with a primary index (or hash key), for instance SSN = '12345' in op1, use the primary index (or hash key) to retrieve the record. This condition retrieves at most a single record.

(d) Using a Primary Index to Retrieve Multiple Records – If the comparison condition is >, >=, <, or <= on a key field with a primary index, for instance DNUMBER > 5 is op2, use the index to find the record satisfying the corresponding equality condition (DNUMBER = 5), then retrieve all subsequent records in the (ordered) file. For the condition DNUMBER < 5, retrieve all the preceding records.

(e) Using a Clustering Index to Retrieve Multiple Records – If the selection condition involves an equality comparison on a non-key attribute with a clustering index, for instance DNO = 5 in op3, use the index to retrieve all the records satisfying the condition.

(f) Using a Secondary (B^+ tree) Index on an Equality Comparison – This search method retrieves a single record if the indexing field is a key or retrieves multiple records if the indexing field is not a key. This can also be used for comparisons involving >, >=, <, or <=.

Search Methods for Complex Selection – If a condition of a SELECT operation is conjunctive condition, i.e., if it is made up of several simple conditions connected with the AND logical connective as in op4, the DBMS uses the following additional methods to implement the operation –

(g) Conjunctive Selection Using an Individual Index – If an attribute involved in any single simple condition is the conjunctive condition has an access path that permits the use of one of the methods (b) to (f), use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive condition.

(h) Conjunctive Selection Using a Composite Index – If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index (or hash structure) exists on the combined fields, for instance, if an index has been created on the composite key (ESSN, PNO) of the WORKS_ON file for op5, we can use the index directly.

(i) Conjunctive Selection by Intersection of Record Pointers – If secondary indexes are available on more than one of the fields involved in simple conditions in the conjunctive condition, and if the indexes include record pointers rather than block pointers, then each index can be used to retrieve the set of record pointers that satisfy the individual condition. The intersection of these sets of record pointers gives the record pointers that satisfy the conjunctive condition, which are then used to retrieve those records directly. If only some of the conditions have secondary indexes, each retrieved record is further tested to determine whether it satisfies the remaining conditions.

(ii) Join Operation – The join operation is the most time consuming operations in query processing. Here, the term join refers to an EQUIJOIN (or NATURAL JOIN). There are many ways to implement a two-way join, which is a join on two files. The algorithms we consider are for join operations of the form

$$R \bowtie_{A=B} S$$

where A and B are domain-compatible attributes of R and S, respectively. The most common techniques for performing such a join, using the following example operations –

(op1) – EMPLOYEE $\bowtie_{DNO=DNUMBER}$ DEPARTMENT

(op2) – DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE

are as follows –

(a) Nested-loop Join (brute force) – For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition t[A] = s[B].

(b) Single-loop Join (using an access structure to retrieve the matching records) – If an index (or hash key) exists for one of the two join attributes, say B of S, retrieve each record t in R, one at a time (single loop), and then use the access structure to retrieve directly all matching records s from S that satisfy s[B] = t[A].

(c) Sort-merge Join – If the records of R and S are physically sorted by value of the join attributes A and B, respectively, we can implement the join in the most efficient way possible. Both files are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B. If the file are not sorted, they may be sorted first by using external sorting. In this method, pairs of file blocks are copied into memory buffers in order and the records of each file are scanned only once each for matching with the other file. Fig. 3.7 shows the sort-merge join algorithm. We use R(i) to refer to the i^{th} record in R. A variation of the sort-merge join can be used when secondary indexes exist on both join attributes. The indexes provide the ability to access the records in order of the join attributes, but the records themselves are physically scattered all over the file blocks, so this method may be quite inefficient, as every record access may involve accessing a different disk block.

```
sort the tuples in R on attribute A; (*assume R has n tuples (records) *)
sort the tuples in S on attribute B; (*assume S has m tuples (records) *)
set i ← 1, j ← 1;
while (i ≤ n) and (j ≤ m)
do{   if R(i)[A] > S(j)[B]
          then      set j ← j+1
      elseif R(i)[A] < S(j)[B]
          then      set i ← i + 1
          else {    (* R(i)[A] = S(j)[B], so we output a matched tuple*)
                    output the combined tuple <R(i), S(j)> to T;
                          (*output other tuples that match R(i), if any*)
                          set l ← j+1;
                          while (l ≤ m) and (R(i)[A] = S(l)[B])
                          do    {           output the combined tuple
                                            <R(i), S(l)> to T; set l ← l+1
                                }
                          (*output other tuples that match S(j), if any*)
                          set k ← i+1;
                          while(k ≤ n) and (R(k)[A] = S(j)[B])
                          do    {           output the combined tuple
                                            <R(k), S(j)>toT; set k ← k+1
                                }
                          set i ← k, j ← l
               }
}
```

Fig. 3.7 Implementing the Operation $T \leftarrow R \bowtie_{A=B} S$

(d) Hash-join – The records of files R and S are both hashed to the same hash file, using the same hashing function on the join attributes A of R and B of S as hash keys. First, a single pass through the file with fewer records (say, R) hashes its records to the hash file buckets, this is called the

partitioning phase, since the records of R are partitioned into the hash buckets. In the second phase, called the ***probing phase,*** a single pass through the other file (S) then hashes each of its records to probe the appropriate bucket, and that record is combined with all matching records from R in that bucket. This simplified description of hash-join assumes that the smaller of the two files fits entirely into memory buckets after the first phase.

(iii) Project Operation – A project operation $\pi_{<\text{attribute list}>}(R)$ is simple to implement if <attribute list> includes a key of relation R, because in this case the result of the operation will have the same number of tuples as R, but with only the values for the attributes in <attribute list> in each tuple. If <attribute list> does not include a key of R, duplicate tuples must be eliminated. This is done by sorting the result of the operation and then eliminating duplicate tuples which appear consecutively after sorting. A sketch of the algorithm is shown in fig. 3.8. Hashing can also be used to eliminate duplicates as each record is hashed and inserted into a bucket of the hash file in memory, it is checked against those already in the bucket. It is a duplicate, it is not inserted.

```
create a tuple t[<attribute list>] in T' for each tuple t in R;
    (*T' contains the projection result before duplicate elimination*)
If <attribute list> includes a key of R
    then T←T'
    else  {    sort the tuples in T';
               set i←1, j←2;
               while i≤n
                    do   {    output the tuple T'[i] to T;
                              while T[i] = T'[j] and j≤n do j←j+1; (*eliminate
                              duplicates*) i←j; j←i+1
                         }
          }
(* T contains the projection result after duplicate elimination *)
```

Fig. 3.8 Implementing the Operation $T \leftarrow \pi_{<\text{attribute list}>}(R)$

Q.46. Describe the steps used to perform JOINS in parallel database.
(R.G.P.V., Dec. 2017)

Ans. The join operation may be parallelized. To join two relations, A and B on the age attribute consider that they are initially distributed across many disks in some way which is not useful for the join operation. It means that the initial partitioning is not depended on the join attribute. The fundamental concept for joining A and B in parallel is to decompose the join into a collection of k smaller joins. Join can be decomposed by partitioning both A and B into a collection of k logical buckets or partitions. The union of the k smaller joins, computes the join of A and B, is confirmed by using the same partitioning function for both A and B. This concept is similar to intuition behind the partitioning phase of a sequential hash join. The partitioning step itself may be done in parallel at these processors

because A and B are initially distributed across several processors. All local tuples at each processor are retrieved and hashed into one of k partitions, with the same hash function employed at all sites, of course. On the other hand, by dividing the range of the join attribute age into k disjoint subranges, A and B can be partitioned. Placing A and B tuples into partitions as per the subrange to which their age values belong. For example, assume that we have 10 processors, the join attribute is age, with values from 0 to 100. Considering uniform distribution, A and B tuples with $0 \le age < 10$ go to processor 1, $10 \le age < 20$ go to processor 2, $20 \le age < 30$ go to processor 3 and so on. This method is more susceptible as compare to hash partitioning to data skew, as long as the subranges are not carefully obtained. Having decided on a partitioning method, each partition to a processor can be assigned, and carry out a local join, by employing any join algorithm at each processor. For such a case, the number of partitions k is selected to be equal to the number of processors n available for performing the join, and each processor transmits tuples in the ith partition to processsor i during partitioning. Each processor joins the A and B tuples assigned to it after partitioning. Each join process receives input A and B tuples from many processors and executes sequential join code. A merge operator merges all incoming A tuples. Another merge operator merges all incoming B tuples. Depending on determining the distribution of the result of the join of A and B, the output of the join process can be break into several data streams. Fig. 3.9 represents the network of operators for parallel join. To simplify the fig. 3.9 it is considered that the processors performing the join are different from the processors which initially haves tuples of A and B and represent only four processors.

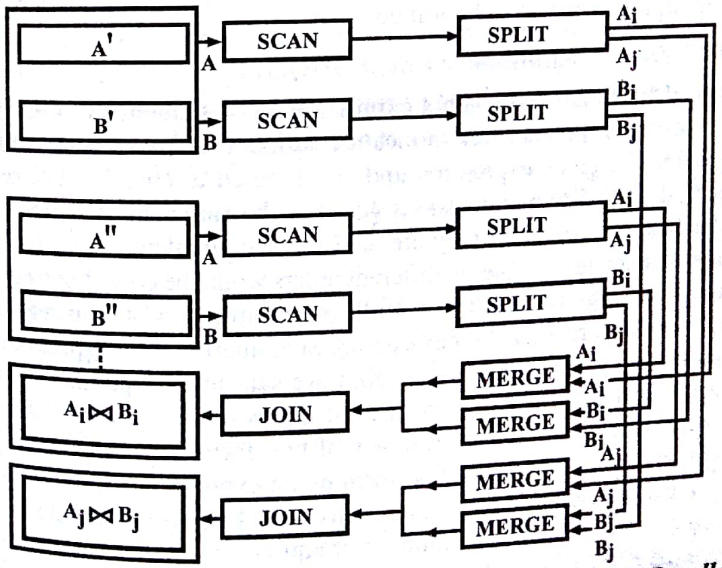


Fig. 3.9 Representation of Data Flow Network of operators for Parallel Join

This algorithm leads to a parallel version of a sort merge join when range partitioning is used, with the profit that the output is available in sorted manner. When hash partitioning is used, a parallel version of a hash join is achieved.

Improved Parallel Hash Join – A hash based refinement of the method provides improved performance. The important observation is that, when A and B are so large and the number of partitions k is selected to be equal to the number of processors n, each partition size can be large, leading to a high cost for each local join at the n processors. On the other hand, execute the smaller joins $A_i \bowtie B_i$, for i = 1 ..... k, one after the other, but with each join executed in parallel using all processors. This method permits to utilize the total available main memory at all n processors in each join $A_i \bowtie B_i$ and is defined in more detail as given below –

(i) At each site, apply a hash function h_1 to partition the A and B tuples at this site into partitions i = 1.....k. Consider A is the smaller relation. The number of partitions k is selected such that every partition of A fits into the aggregate or combined memory of all n processors.

(ii) Process the join of the ith partitions of A and B for i = 1 to k. For calculating $A_i \bowtie B_i$, do the following at every site –

(a) Apply a second hash-function h_2 to all A_i tuples to compute in unich they must be joined and transmit tuple t to sile $h_2(t)$.

(b) As A_i tuples arrive to be joined, combine them to an in-memory hash table.

(c) Apply h_2 to B_i tuples to determine where they should be joined and transmit tuple t to site h_2 (t) after all A_i tuples have been distributed.

(d) Probe the in-memory table of A_i tuples and output result tuples, as B_i tuples arrive to be joined.

Q.47. Discuss equivalence rule in brief.

Ans. Every relational algebra expression has a sequence of operations. A query processing strategy determines a relational algebra expression that is equivalent to the given expression and is estimated to cost less to execute. In relational algebra if two expressions generate the same set of tuples on every legal database instance that they are said to be equivalent. Note that the two expressions generate the tuples in different orders would be considered equivalent. The inputs and outputs, in SQL, are multisets of tuples, and a multiset version of the relational algebra is used for evaluating SQL queries. Two expressions in the multiset version of the relational algebra are said to be equivalent if the two expressions generate the same multiset of tuples on every legal database. An equivalence rule refers that expressions of two forms are equivalent. We can transform an expression of the first form by an expression of the second form or vice versa because, the two expressions would generate the same result on any legal database. There are a number of equivalence rules that we can use to transform an expression into an equivalent one. We use these rules to generate systemalically all expressions equivalent to the given query.

Q.48. How to estimate statistics of expression results ?

Ans. The rules that the database system chooses for evaluating an operation depends on the size of each relation and on the distribution of values within columns. Given an expression $m \bowtie (p \bowtie q)$ to estimate the cost of joining m with $(p \bowtie q)$, we need to have estimates of statistics like $p \bowtie q$. Some statistics about database relations that are stored in database system catalogs includes –

(i) The number of tuples in the relation r.

(ii) The size of a record (tuple) of relation r in bytes.

(iii) The number of distinct values that appear in the relation r for a particular attribute.

(iv) The number of block containing tuples of relation r.

These statistics enable us to estimate the sizes of the results of various operations and the cost of executing the operations. Statistical information about relations is particularly useful when several indices are available to assist in the processing of a query. The presence of these structures has a significant influence on the choice of a query processing strategy.

If we want to maintain accurate statistics, then every time a relation is modified, we must also update the statistics. This update creates a substantial amount of overhead. That's why, systems do not update the statistics on every modification. Instead, they update the statistics during period of light system load. Thus, the statistics used for selecting a query processing strategy may not be completely accurate. Although, the statistics will be sufficiently accurate to provide a good estimation of the relative costs of the different plans if not too may updates occur in the intervals between the updates of the statistics.

Q.49. How does a DBMS represent a relational query evaluation plan ?

Ans. An execution (evaluation) plan for a relational algebra expression represented as a query tree includes information about the access methods available for each relation as well as the algorithms to be used in computing the relational operators represented in the tree. As a simple example, consider query Q_1 as given below –

$$\text{RESEARCH_DEPT} \leftarrow \sigma_{\text{DNAME='RESEARCH'}} (\text{DEPARTMENT})$$

$$\text{RESEARCH_EMPS} \leftarrow (\text{RESEARCH_DEPT} \bowtie_{\text{DNUMBER=DNO}} \text{EMPLOYEE})$$

$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, ADDRESS}} (\text{RESEARCH_EMPS})$$

whose corresponding relation algebra expression is

$$\pi_{\text{FNAME, LNAME, ADDRESS}} (\sigma_{\text{DNAME = 'RESEARCH'}} (\text{DEPARTMENT}) \bowtie_{\text{DNUMBER=DNO}} \text{EMPLOYEE})$$

The query tree is shown in fig. 3.10. To convert this into an execution plan, the optimizer might choose an index search for the SELECT operation (assuming one exists), a table scan as access method for EMPLOYEE, a nested-loop join algorithm for the join, and a scan of the JOIN result for the PROJECT operator. In addition, the approach taken for executing the query may specify a materialized or a pipelined evaluation.

With ***materialized evaluation,*** the result of an operation is stored as a temporary relation (that is, the result is *physically materialized*). For example, the join operation can be computed and the entire result stored as a temporary relation, which is then read as input by the algorithm that computes the PROJECT operation, which would produce the query result table. On the other hand, with ***pipelined evaluation,*** as the resulting tuples of an operation are produced, they are forwarded directly to the next operation in the query sequence. For example, as the selected tuples from DEPARTMENT are produced by the SELECT operation, they are placed in a buffer; the JOIN operation algorithm would then consume the tuples from the buffer, and those tuples that result from the JOIN operation are pipelined to the projection operation algorithm. The advantage of pipelining is the cost savings in not having to write the intermediate results to disk and not having to read them back for the next operation.

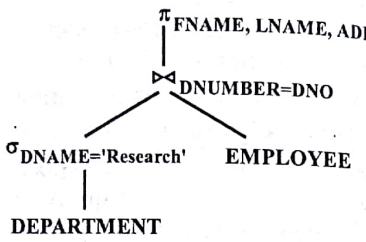


Fig. 3.10 A Query Tree for Query Q_1

CASE STUDY OF ORACLE AND DB2

Q.50. What do you mean by Oracle ? Give its architecture.

Ans. Oracle is one of the most popular relational database management system (RDBMS). It is based on the client/server architecture. A client sends the requests, such as connection to the database or retrieving data from the database, for several operations. A server accepts, processes and returns the results of the request, such as the data requested by the client, back to the client. Oracle provides the software for both the server and the client.

Oracle's popularity is based on large measures of exceptional performance and scalability for users, applications and data.

Oracle is organized into a central, kernel part and variety of interfaced tools that make use of it. The general organization of the product is represented conceptually in fig. 3.11.

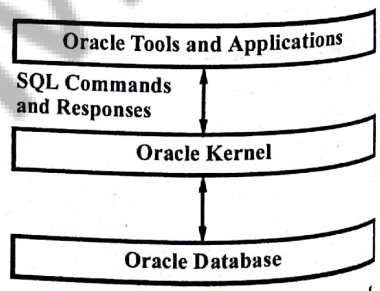


Fig. 3.11 ORACLE Conceptual Structure

The kernel part of the structure is responsible for the definition and storage of data, database access control and concurrent access handling, backup and recovery of the database and interpretation of SQL commands. Although some of the user interfaces provided by the ORACLE tools do not require the use of SQL on the part of their users, the interface between all tools and the kernel is driven by SQL commands and the responses the kernel makes to them. ORACLE provides full support for communications networking and distributed databases with its SQL*Net and SQL*Star products.

The architecture of Oracle comprises of database and instance. An Oracle database is a collection of files having data and programs to manage the data. The instance comprises of the memory structures and processes. The memory structures are areas of the computer's memory that are used to store the data of the database. The processes operate on the database and memory structures to perform various database operations such as fetching data from the database and storing it in the memory structures. Most of these processes are referred to as the background processes. The memory area allocated for an Oracle instance is called SGA or System Global Area.

Q.51. Write short note on DB2. ***(R.G.P.V., Dec. 2017)***

Ans. Since the 1970s, when IBM research invented the relational model and the Structured Query Language (SQL), IBM has developed a complete family of data servers. Development started on mainframe platform such as Virtual machine (VM), Virtual Storage Extended (VSE), and Multiple Virtual Storage (MVS). In 1983, DB2 for MVS Version 1 was born. "DB2" was used to indicate a shift from hierarchical databases – such as the Information Management System (IMS) popular at the time – to the new relational databses. DB2 development continued on mainframe platforms as well as on distributed platforms. Fig. 3.12 shows some of the highlights of DB2 history.

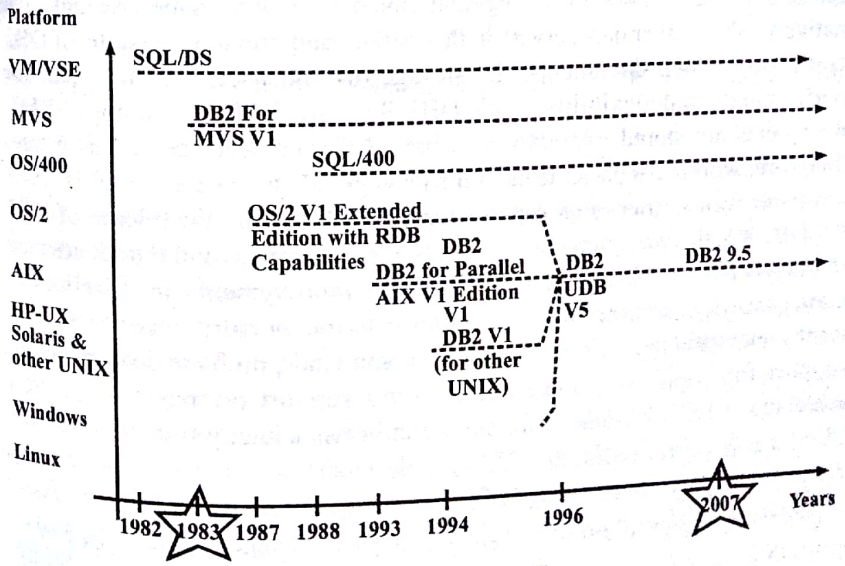


Fig. 3.12 DB2 Timeline

In 1996, IBM announced DB2 universal database (UDB) version 5 for distributed platforms. With this version, DB2 was able to store all kinds of electronic data, including traditional relational data, as well as audio, video, and text documents. It was the first version optimized for the Web, and it supported a range of distributed platforms – for example, OS/2, Windows, AIX, HP-UX, and Solaris – from multiple vendors. Moreover, this universal database was able to run on a variety of hardware, from uniprocessor systems and symmetric multiprocessor (SMP) systems to massively parallel processing (MPP) systems and clusters of SMP systems.

Even though the relational model to store data is the most prevalent in the industry today, the hierarchical model never lost its importance. In the past few years, due to the popularity of Extensible Markup Language (XML), a resurgence in the use of the hierarchical model has taken place. XML, a flexible, self-describing language, relies on the hierarchical model to store data. With the emergence of new Web technologies, the need to store unstructured types of data, and to share and exchange information between businesses, XML proves to be the best language to meet these needs. Today we see an exponential growth of XML documents usage.

IBM recognized early on the importance of XML, and large investments were made to deliver pureXML technology; a technology that provides for better support to store XML documents in DB2. After five years of development, the effort of 750 developers, architects, and engineers paid off with the release of the first hybrid data server in the market – DB2 9.DB2 9, available since July 2006, is a hybrid (also known as multi-structured) data server because it allows for storing relational data, as well as hierarchical data, natively. While other data servers in the market, and previous versions of DB2 could store XML documents, the storage method used was not ideal for performance and flexibility. With DB2 9's pure XML technology, XML documents are stored internally in a parsed hierarchical manner, as a tree; therefore, working with XML documents is greatly enhanced. In 2007, IBM has gone even further in its support for pure XML, with the release of DB2 9.5.DB2 9.5, the latest version of DB2, not only enhances and introduces new features of pure XML, but it also brings improvements in installation, manageability, administration, scalability and performance, workload management and monitoring, regulatory compliance, problem determination, support for application development, and support for business partner applications. DB2 is available for many platforms including system z (DB2 for z/OS) and System i (DB2 for i5/OS). Unless otherwise noted, when we use the term DB2, we are referring to DB2 version 9.5 running on Linux, UNIX, or Windows. DB2 is part of the IBM information management (IM) portfolio.

⌘⌘

UNIT 4

RELATIONAL DATABASE DESIGN – FUNCTIONAL DEPENDENCY – DEFINITION, TRIVIAL AND NON-TRIVIAL FD, CLOSURE OF FD SET, CLOSURE OF ATTRIBUTES, IRREDUCIBLE SET OF FD

Q.1. What is functional dependency ? Explain the role of FD in the process of normalization. ***(R.G.P.V., June 2010)***

Or

With the help of an example relation and the dependency diagram, explain the concept of functional dependency. ***(R.G.P.V., Dec. 2012)***

Or

Write a note on functional dependencies. ***(R.G.P.V., Dec. 2014)***

Or

Define the term functional dependency. ***(R.G.P.V., Dec. 2011, 2013)***

Or

Write short note on functional dependency. ***(R.G.P.V., Dec. 2016)***

Ans. A functional dependency, (or FD) denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the tuples that can form a relation state of r of R. The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$, we must have $t_1[Y] = t_2[Y]$. It means that the values of the Y of a tuple in r depend on, or are determined by, the values of the X component. Alternatively, the values of X component of a tuple uniquely (or functionally) determine the values of the Y component. In other words, there is a functional dependency from X to Y or that Y is functionally dependent on X. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side. Thus, X functionally determines Y in a relation schema R if and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y-value.

It should be noted that

(i) If a constraint on R states that there cannot be more than one tuple with a given X-value in any relation i.e., X is a candidate key of R. This implies that $X \rightarrow Y$ for any subset of attributes Y of R.

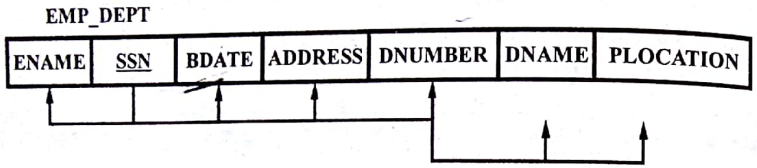
(ii) If $X \rightarrow Y$ in R, this does not say whether or not $Y \rightarrow X$ in R.

For example, in the relation schema EMP-PROJ in fig. 4.1, the following functional dependencies should hold –

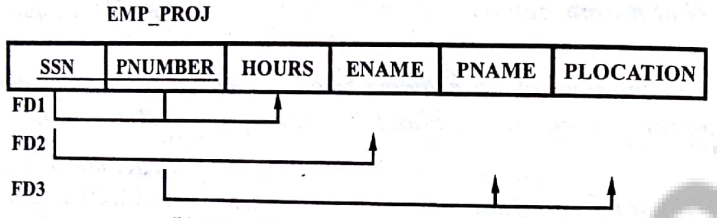
(i) SSN → ENAME

(ii) PNUMBER → {PNAME, PLOCATION}

(iii) {SSN, PNUMBER} → HOURS



(a) The EMP_DEPT Relation Schema



(b) The EMP_Relation Schema

Fig. 4.1 Two Relation Schemas and their Functional Dependencies; Both Surface Update Anomalies

These functional dependencies specify that (i) the value of an employee's social security number (SSN) uniquely determines the employee name (ENAME), (ii) the value of a project's number (PNUMBER) uniquely determines that project name (PNAME) and location (PLOCATION) and (iii) a combination of SSN and PNUMBER values uniquely determines the number of hours the employee works on the project per week (HOURS). Alternatively, we can say that ENAME is functionally determined by SSN, or "given a value of SSN, we know the value of ENAME", and so on.

Q.2. What are the uses of functional dependencies ? (R.G.P.V., Dec. 2015)

Ans. We use functional dependencies –

(i) To test relations to see if they are equal under a given set of functional dependencies. If a relation r is legal under a set F of functional dependencies, we say that r satisfies F.

(ii) Specify constraints on the set of legal relations; we set that F holds on R if all legal relations on R satisfy the set of functional dependencies F.

Q.3. State Armstrong's axioms of functional dependency.
(R.G.P.V., June 2014)

Or

What are the inference rules for functional dependencies ?
(R.G.P.V., June 2016)

Or

What are the inference rules for multivalued dependencies ? Explain.
(R.G.P.V., Dec. 2016)

Ans. The set of inference rules (also called Armstrong's axioms) by which new FD's can be inferred from given FD's are as follows –

(i) Reflexivity – If α is a set of attributes and , $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ holds.

(ii) Augmentation – If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.

(iii) Transitivity – If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.

Several other rules derived from the three given rules can be as follows –

(iv) Union – If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.

(v) Decomposition – If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.

(vi) Pseudotransitivity – If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.

Q.4. Define a functional dependency list and discuss the six inference rules for functional dependencies, give relevant example. (R.G.P.V., Dec. 2014)

Ans. Refer to Q.1 and Q.3.

Q.5. Explain trivial and non-trivial dependencies. (R.G.P.V., June 2016)

Ans. Some functional dependencies are said to be ***trivial,*** because they are satisfied by all relations. For example, $A \rightarrow A$ is satisfied by all relations involving attribute A. By the definition of functional dependency, we see that, for all types t_1 and t_2 such that $t_1[A] = t_2[A]$, it is the case that $t_1[A] = t_2[A]$.

Similarly, $AB \rightarrow A$ is satisfied by all relations involving attribute A.

In fact, FD is trivial if and only if the right-hand side is a subset of the left-hand side. Non-trivial dependencies are one that is not trivial. That shows the real relations between the attributes of the relation schema. In general, the functional dependency of the form $\alpha \rightarrow \beta$ is nontrivial if and only if $\beta \not\subseteq \alpha$. For example, consider the relation –

Product (P#, Pname, Color, Weight).

In this relation, we have following dependencies –

P# → Pname

P# → Color

These are non-trivial in nature, as they are not showing dependencies among itself.

Uniweb

Q.6. What are the different types of functional dependencies ?

Ans. There are many types of functional dependencies that depending on different criteria –

(i) Full Functional Dependency – For a relation schema R and a FD, $A \rightarrow B$, B is fully functionally dependent on A if there is no C, where C is proper subset of A, such that $C \rightarrow B$. The dependency $A \rightarrow B$ is left reduced, that is, there is no extraneous attributes in the left hand side in this dependency.

(ii) Partial Dependency – A partial dependency is one in which an attribute is functionally dependent on only a part of a multi-attribute primary key. Given a relation schema R with the functional dependency F defined on the attributes of R and K as the candidate key, if X is a proper subset of K and if $|F| = X \rightarrow A$, then A is said to be partially dependent on K. It is associated with 1 NF.

(iii) Transitive Dependency – Let a relation R with the functional dependency F defined on the attributes of R, let A and B be subsets of R and let X be the attribute of R such that X is not subset of B, and X is not subset of AB. If the set of functional dependencies $\{A \rightarrow B, B \rightarrow X\}$ is simplified by F, then X is transitively dependent on A.

Suppose, we have

$$S\# \rightarrow City$$
$$City \rightarrow Status$$

Here, transitive dependency that exists is

$$S\# \rightarrow Status$$

(iv) Trivial and Non-trivial Dependencies – Refer to Q.5.

Q.7. What is partial dependency ? With which normal form is it associated ? ***(R.G.P.V., June 2016)***

Ans. Refer to Q.6.

Q.8. Explain functional dependency and trivial functional dependency by taking examples. List all functional dependencies satisfied by the relation of figure given below –

A	*B*	*C*
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_3	c_1
a_2	b_3	c_3

(R.G.P.V., June 2013)

Ans. **Functional Dependency** – Refer to Q.1.

Trivial Functional Dependency – Refer to Q.5.

The functional dependencies satisfied by the relation of above figure are

$A \rightarrow B$, $B \rightarrow A$, $AC \rightarrow B$ and $BC \rightarrow A$.

Q.9. Write short note on closure of a set of functional dependencies.

Ans. If be a consider the given set of functional dependencies, that is not sufficient then we need to determine all functional dependencies that hold. Given a set of F (functional dependencies) it can be prove that certain other functional dependencies hold. Such types of functional dependencies are "logically implied by F.

Formally, given a relation R, a functional dependency f on R is logically implied by a set of functional dependencies F on R if every relation instance r (R) that satisfies F also satisfies f.

Thus, the set of functional dependencies that is logically implied by F is called the closure of F and is written as F^+.

Definition – If F is a set of FDs on a relation schema R, then F^+, the closure of F, is the smallest set of FDs such that $F^+ \supseteq F$ and no FD can be derived from F by using the inference axioms that are not contained in F^+.

For example, we are given a relation schema R = (L, M, N, O, P, Q) and the set of functional dependencies

$$L \rightarrow M$$
$$L \rightarrow N$$
$$NO \rightarrow P$$
$$NO \rightarrow Q$$
$$M \rightarrow P$$

The functional dependency $L \rightarrow P$ is logically implied. That is, whenever our given set of functional dependencies holds on a relation, $L \rightarrow P$ must also hold on the relation. Suppose that t_1 and t_2 are tuples such that

$$t_1 [L] = t_2 [L]$$

Since we are given that $L \rightarrow M$, it follows from the definition of functional dependency that

$$t_1 [M] = t_2 [M]$$

Then, since we are given that $M \rightarrow P$, it follows from the definition of functional dependency that

$$t_1 [P] = t_2 [P]$$

Therefore, we have shown that, whenever t_1 and t_2 are tuples such that $t_1 [L] = t_2 [L]$, it must be that $t_1 [P] = t_2 [P]$.

Q.10. Define canonical cover. ***(R.G.P.V., Dec. 2015)***

Ans. A set of functional dependencies F_c is a ***canonical*** cover if every FD in F_c satisfies the following –

(i) Each FD in F_c is simple. Recall that in a simple FD the right-hand side has a single attribute i.e., each FD is of the form $X \rightarrow A$.

MSUniversity

(ii) For no FD $X \to A$ with $Z \subset X$ is $\{(F_c - (X \to A)) \cup (Z \to A)\}$ F_c. In other words, the left-hand side of each FD does not have any extraneous attributes, or the FDs in F_c are left reduced.

(iii) No FD $X \to A$ is redundant i.e., $\{F_c - (X \to A)\}$ does not logically imply F_c.

A canonical cover is sometimes called ***minimal***.

Q.11. Discuss the term closure of a set of attributes.

Ans. An algorithm is discovered for computing the set of attributes functionally specified by α, where α is a superkey. The closure of a set of attribute provides way to do this is to compute F^+, and it is take all functional dependencies with α as the left-hand side, and its take the union of the right-hand side of all such dependencies.

where, α be a set of attributes, the set of all attributes functionally specified by α under a set F of functional dependencies is called the closure of α under F. It is denoted by α^+. Fig. 4.2 shows an algorithm in pseudocode to compute α^+. The input is a set F of functional dependencies and the set α of attributes. The output is stored in the variable result.

```
result := α;
while (changes to result) do
    for each functional dependency
            β → γ in F do
    begin
        if β ≤ result then
            result : = result Uγ;
    end
```

Fig. 4.2 An Algorithm to Compute α^+

Q.12. Define irreducible sets of dependencies. ***(R.G.P.V., Dec. 2014)***

Ans. A set S of FDs is irreducible if and only if it satisfies the following three properties –

(i) The right-hand side (the dependent) of every FD in S involves just one attribute (i.e., it is a singleton set).

(ii) The left-hand side (the determinant) of every FD in S is irreducible in turn–meaning that no attribute can be discarded from the determinant without changing the closure S^+(i.e., without converting S into some set not equivalent to S). We will say that such an FD is left-irreducible.

(iii) No FD in S can be discarded from S without changing the closure S^+ (i.e., without converting S into some set not equivalent to S).

With regard to points (ii) and (iii), it is not necessary to know exactly what the closure S^+ is in order to tell whether it will be changed if something is discarded.

Q.13. Given below are two sets of FDs for a relation R(A, B, C, D, E). Are they equivalent ?

(i) $A \to B, AB \to C, D \to AC, D \to E$

(ii) $A \to BC, D \to AE$

(R.G.P.V., June 2017)

Ans. Taking functional dependency (i)

$A \to B, AB \to C, D \to AC$ and $D \to E$

Since $A \to B$

So, putting A in place of B in $AB \to C$

$\Rightarrow AA \to C$

$\Rightarrow A \to C$

Now, since $A \to B$ and $A \to C$.

$\therefore$ **$A \to BC$** (Union rule of FD)

Now, since $D \to AC$

$\therefore$ $D \to A$ and $D \to C$ (Decomposition rule of FD)

Now, we have

$D \to A$ and $D \to E$

$\therefore$ **$D \to AE$** (Union rule of FD)

Now, since $D \to A$ and $A \to C$ can easily determine $D \to C$ (By transitivity rule)

So, there is no need to express $D \to C$

Hence, the resultant functional dependency is –

$A \to BC, D \to AE$

which is same as functional dependency (ii)

Hence, the FDs (i) and (ii) both are equivalent.

Q.14. Compute the closure of the following set F of functional dependencies for relation schema –

$$R = (A, B, C, D, E)$$
$$A \to BC$$
$$CD \to E$$
$$B \to D$$
$$E \to A$$

List candidate keys for R. Compute B^+ and the canonical cover F_C ***(R.G.P.V., June 2007)***

Or

Give closure of the following set F of functional dependencies for the relation R = (A, B, C, D, E) where

$$F = \{A \to BC$$
$$B \to D$$
$$CD \to E$$
$$E \to A\}$$

Also list all candidate keys of R. ***(R.G.P.V., June 2013)***

Ans. The members of closure of F, F^+ are as follows –

(i) $A \to CD$. Since by augmentation rule on $B \to D$ to infer $BC \to CD$. Now, applying transitivity on $A \to BC$ and $BC \to CD$.

(ii) BC → E. Since B → D and CD → E, the pseudotransivity rule implies BC → E.

(iii) A → B and A → C. Since, A → BC holds, applying decomposition rule we get A → B and A → C.

The candidate keys are CD and BC.

The canonical cover F_c is {A → BC, B → D, E → A, C → E}.

NORMALIZATION – 1NF, 2NF, 3NF

Q.15. Justify the need for normalization with examples.
(R.G.P.V., Dec. 2012)

Or

Specify the need of normalization. ***(R.G.P.V., June 2014)***

Or

What do you mean by normalization ? ***(R.G.P.V., June 2015)***

Ans. Normalization is a process of converting a relation into a standard form. The normalization process or just normalization is built around the concept of normal forms. A relation is said to be in a particular normal form if it satisfies a certain set of conditions. For example, let we have a relation suppliers with attributes name, address, item, price.

suppliers (name, address, item, price)

Now, the problems that we can face due to this relation are –

(i) Redundancy – The address of supplier is repeated once for each item supplied. So this causes repetition of same information many times.

(ii) Potential Inconsistency (Update Anomalies) – As we have redundancy in our relation i.e., same information is repeated many times. It is possible that when we are updating some information, some get updated while others remain fixed, which will result inconsistency. As in above example, if we update the address of a supplier in one tuple, while leaving it fixed in another. Thus, we would not have a unique address for each supplier.

(iii) Insertion Anomalies – We cannot insert a tuple in a relation, if that tuple does not have value for any of the attribute in relation.

For example, we cannot record an address for a supplier, if that supplier does not currently supply at least one item. We might put null values in the item and price components of a tuple for that supplier but then, when enter an item for that supplier, we will remember to delete the tuple with the nulls.

(iv) Deletion Anomalies – On deleting some information, we lose some other information too. For example, if we delete all items supplied by one suppliers, we unintentionally lose track of its address.

So, due to the all of the above problems we normalize a relation. In this example, if we replace suppliers by two relation schemas

SA (name, address)

SIP (name, item, price)

Then in SA, for each supplier we have an address separately without any redundancy. At the same time, we can enter address for a supplier even if it currently supplies no items.

Q.16. What is normalization ? Explain the normalization techniques using functional dependencies with relevant examples. (R.G.P.V., June 2011)

Ans. Refer to Q.15 and Q.1.

Q.17. Write the purpose of normalization in DBMS.
(R.G.P.V., June 2016)

Ans. The objectives of normalization process are –

(i) To make it feasible to represent any relation in the database.

(ii) To obtain powerful relational retrieval algorithms based on a collection of primitive relational operators.

(iii) To free relations from undesirable insertion, update and deletion anomalies.

(iv) To reduce the need for restructuring the relations as new data types are introduced.

Q.18. What is first normal form ? ***(R.G.P.V., Dec. 2015)***

Ans. A relation is in 1NF if and only if, for every legal value of that relation, every tuple has exactly one value for each attribute.

Or

A relation is said to be in first normal form (1NF) if the values in the domain of each attribute of the relation are atomic. (A domain is atomic if the elements of domain are considered to be indivisible units.) In other words, only one value is associated with each attribute and the value is not a set of values or a list of values. A database is in first normal form if every relation included in database is in 1NF.

Q.19. How do we achieve 1 NF ?

Or

Describe the characteristics of a table that is not normalized form. Describe how such a table converted to a first normal form relation.
(R.G.P.V., June 2011)

Ans. First normal form deals only with the basic structures of the relation and does not resolve the problems of redundant information or any other anomalies. Let us take an example of an organization, where we have information about employee_id, employee_name, phone_no and their salary.

Vtuinsider

Table 4.1 Non-domain Form

employee_id	*employee_name*	*phone_no*	*salary*
101	Priya	432049	10,000
102	Ashish	420420 323232	15,000
103	Kaushlendra	583119	12,000
104	Piyush	500698 434596	20,000

Here, we see that an employee can have more than one phone number. So, each tuple of above table does not have only one value for each attribute.

For converting it into 1NF we use the following steps –

(i) For each repeating field value, create a new tuple.

(ii) Insert the repeating field value in that field of the new tuple.

(iii) Duplicate the values of all other attributes of the original tuple and put them in the new tuple.

In our example, we have phone_no field of Ashish and Piyush have repeating value. So, we have to create new tuples only for these two employees.

This can be done as follows –

Table 4.2 Normalized-form

employee_id	*employee_name*	*phone_no*	*salary*
101	Priya	432049	10,000
102	Ashish	420420	15,000
102	Ashish	323232	15,000
103	Kaushlendra	583119	12,000
104	Piyush	500698	20,000
104	Piyush	434596	20,000

Q.20. What are the anomalies associated with 1 NF ?

Ans. Problems or anomalies to be faced in 1NF are as follows –

(i) INSERT – We cannot insert the information about an employee, who does not have any phone_no.

(ii) DELETE – If we delete the sole first tuple for a particular employee, we delete not only the phone_no of employee but also lose other information, such as salary and employee_id. For example, if we delete a tuple of employee Priya whose phone is disconnected, then we also lose other information of Priya such as salary. (Insert and delete are two sides of same coin).

(iii) UPDATE – In this form of relation, we have same information repeated many times. If we want to update some information, for this first we have to search all the tuples containing that information and then update all that tuples. For example, if salary of Piyush get incremented to 25,000 and we

want to update this information in our relation, for this first of all we have to search all the tuples having employee_name "Piyush" and then update the salary to 25,000 of each tuple. But if we forget updation of any of tuple named Piyush then it will result inconsistency. To avoid this, we go for 2NF.

Q.21. What is 2 NF ?

Ans. A relation is in second normal form (2NF), if it is in 1NF and if all nonprime attributes are fully functionally dependent on the primary key.

A database schema is in 2NF if every relation included in the database is in 2NF.

Q.22. What are the advantages of 2 NF over 1 NF ?

Ans. The advantage of 2NF is that, the problems that we were facing in 1NF are solved in 2NF. These are solved as follows –

(i) Insert – We can insert the information that supplier S lives in city Paris, eventhough S1 does not currently supply any part, by simply inserting appropriate tuple into relation S.

(ii) Delete – We can delete the connection S1 and P2 by deleting a tuple from relation SP, without losing the information that S1 is located in Paris, as this information we have in relation S.

(iii) Update – The city for a given supplier is written once, its not repeated many times as shown in relation S. The primary key of this relation is S#. For one value of S#, there is only one value of city, which was repeated many times in 1NF.

So, if we want to update the city of a particular supplier, updation is required only once. Eventhough, 2NF faces some anomalies for removal of which we go for 3NF. These anomalies are as follows –

(i) Insert – We cannot insert the fact that a particular city has a particular status e.g., we cannot state that a supplier in Italy must have a status 30, until we have some supplier located in that city.

(ii) Delete – If we delete the sole S tuple for a particular city, we delete not only the information for the supplier concerned, but also the information that city has that particular status. For example, if we delete second tuple from relation S. We lose the information that city London has status 20. At the same time, information that, supplier S2 is concerned with city London.

(iii) Update – Status of a city in relation S appears repeatedly. For example, in relation S status of city Paris appears repeatedly. For updating the status of city Paris, first we have to find all the tuples that have city Paris then we have to update them.

If we miss any tuple, it will result inconsistency.

Q.23. How do we achieve 2 NF ?

Ans. The steps that involved in transforming the relation from 1NF to 2NF are as follows –

(i) Identify the set of attributes that makes up the primary key.

(ii) Create all subsets of the above set obtained in step (i).

(iii) Designate each of these subsets at the primary key of a relation that contains those attributes which are dependent on these primary keys.

For example, we take a relation of marketing department named sale with the attributes as shown in table 4.3.

Table 4.3 Relation Sale

S#	Status	City	P#	Pname	Color	Qty
S1	10	Paris	P1	Bulb	Pink	300
S1	10	Paris	P2	Tubes	Yellow	100
S2	20	London	P3	Fan	Green	500
S3	30	Italy	P4	Cooler	Black	200
S3	30	Italy	P1	Bulb	Pink	450
S4	40	India	P5	Lamp	White	400
S5	10	Paris	P2	Tubes	Yellow	250

The explanation of the attributes of table 4.3 is as follows –

(i) ***S#*** – Supplier number.

(ii) ***Status*** – This denotes status of supplier with constraint that CITY determines status.

(iii) ***City*** – Name of city in which supplier lives.

(iv) ***P#*** – Product number.

(v) ***Pname*** – Name of the product.

(vi) ***Color*** – Color of the product.

(vii) ***Qty*** – Denotes quantity of a product sold by a particular supplier.

This relation is already in 1NF and faces all the anomalies that 1NF relation faces i.e., insertion, deletion and updation. After 1NF, for simplicity we draw the functional dependency diagram (FD diagram) of relation, as shown in fig. 4.3.

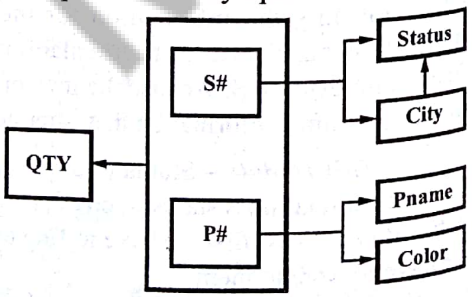


Fig. 4.3 FD for Relation Sale

To convert this relation sale in 2NF, the steps are as follows –

(i) Attributes that make primary key are S# and P# i.e., {S#,P#}.

(ii) Subsets of above set are {}, {S#}, {P#}, {S#, P#}.

(iii) We designate these subsets as the primary key of the relation with attributes that are dependent on these primary key.

So, we have one relation S with primary key S# and attributes Status and City. Another relation P with primary key P# and attributes Pname and Color.

Another relation SP with primary key S# and P# with attribute Qty.

In second normal form, above relation can be given as in fig. 4.4.

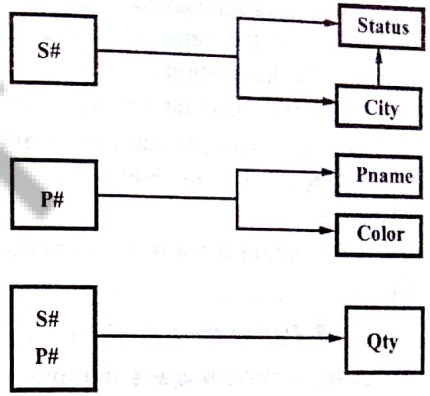


Fig. 4.4 FD Diagram in 2NF

These relations can be given in following way –

Table 4.4 Sample Values for Relation S, P and SP

(a) Relation S

S#	Status	City
S1	10	Paris
S2	20	London
S3	30	Italy
S4	40	India
S5	10	Paris

(b) Relation SP

S#	P#	Qty
S1	P1	300
S1	P2	100
S2	P3	500
S3	P4	200
S3	P1	450
S4	P5	400
S5	P2	250

(c) Relation P

P#	Status	City
P1	Bulb	Pink
P2	Tubes	Yellow
P3	Fan	Green
P4	Cooler	Black
P5	Lamp	White

Q.24. Consider the relation r (A, B, C, D, E) and the set F = {AB → CE, E → AB, C → D}, what is the highest normal form of this relation. *(R.G.P.V., Dec. 2013)*

Ans. The given relation is already is 1 NF, because r is a relation by definition.

We need to determine all possible keys of the relation to find the normal form of the given relation. The two possible keys of this relation are AB and E. Let us check why it is so.

AB is a key.

(i) AB → A (Reflexivity axiom)

(ii) AB → B (Reflexivity axiom)

(iii) From AB → CE (given) we have that AB → C (projectivity axiom)

(iv) From AB → CE (given) we have that
AB → E (projectivity axiom)

MsinVenter

(v) From AB → C (step (iii)) and C → D (given) we have that AB → D (Transitivity axiom) since AB determines all attributes of the relation, AB is a key.

E is a key

Since E → AB (given) and AB is a key then E is also a key because by application of the transitivity axiom E can functionally determine all other attributes of the relation.

The prime attributes are A, B and E.

The nonprime attributes are C and D.

The relation is in 2 NF because there are no partial dependencies on any of the key.

The relation is not in 3 NF because there is a transitivity dependency on the keys.

Q.25. Define the term third normal form. (R.G.P.V., Dec. 2011, 2013)

Ans. A relation is in third normal form (3NF) if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key. By the transitive functional dependency, we mean that –

If, x (primary key) → y and y → z then x → z.

So, z is transitively dependent on x. And according to definition of 3NF as such dependency should be there, that is relation.

Q.26. What are the advantages of 3NF over 2 NF ?

Ans. The advantage is that we can solve the problems that we were facing in 2NF. We can solve them as follows –

(i) INSERT – In relation CS, we can insert a tuple indicating that a particular city has a particular status. So, we can say that supplier in Italy must have status 30.

(ii) DELETE – On deleting any tuple from relation SC, we do not lose information about the status of city in which that supplier lives. For example, if we delete second tuple from relation SC, then we also know that city London has status 20.

(iii) UPDATE – Similarly to insertion and deletion, updation is also easy. If we want to change the status of any city then we have to just update one tuple in relation CS. For example, if we wish to change the status of London from 20 to 50 then we have to just update the second tuple in relation CS. Even now problem is not fully solved. For this, we go for further normalization.

Q.27. How do we achieve 3NF ?

Ans. Steps involved in transforming the relation from 2NF to 3NF.

(i) Determine the non-key attributes that determine some other non-key attributes.

(ii) It make separate relation taking first one as primary key.

Or

(i) In other words, we can say that we try to find attributes that show transitive dependency between them. This we can search from FD diagram.

(ii) Try to avoid this transitivity by using steps given above.

Let us consider the fig. 4.5.

In our example from fig. 4.5 we can say that in relation S –

S# → City

City → Status

So, S# → Status.

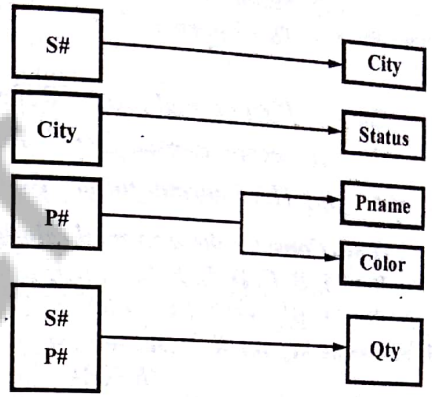


Fig. 4.5 FD Diagram in 3NF

This shows transitive dependency, which we have to remove other relationship P and SP does not show any such type of dependency.

So following the step (i), we determine the non-key attribute that determines the some other non-key attribute. City that is a non-key attribute determines status which is a non-key attribute i.e., for any two equal values of city, status definitely have equal values. The relation can be given as in table 4.5.

Table 4.5 Relations in 3NF

(a) Relation SC

S#	*City*
S1	Paris
S2	London
S3	Italy
S4	India
S5	Paris

(b) Relation CS

City	*Status*
Paris	10
London	20
Italy	30
India	40

(c) Relation SP

S#	*P#*	*Qty*
S1	P1	300
S1	P2	100
S2	P3	500
S3	P4	200
S3	P1	450
S4	P5	400
S5	P2	250

(d) Relation P

P#	*Pname*	*Color*
P1	Bulb	Pink
P2	Tubes	Yellow
P3	Fan	Green
P4	Cooler	Black
P5	Lamp	White

Q.28. What is normalization ? Why is it required ? Explain 1 NF, 2NF and 3NF with suitable example. ***(R.G.P.V., May 2018)***

Ans. **Normalization and it Required** – Refer to Q.15.

1 NF with Example – Refer to Q.18 and Q.19.

2 NF with Example – Refer to Q.21 and Q.23.

3 NF with Example – Refer to Q.25 and Q.27.

Visinvent

Q.29. Explain each of the following with example –

(i) First normal form

(ii) Second normal form

(iii) Third normal form.

(R.G.P.V., June 2017)

Ans. (i) First normal form – Refer to Q.18 and Q.19.

(ii) Second normal form – Refer to Q.21 and Q.23.

(iii) Third normal form – Refer to Q.25 and Q.27.

Q.30. Consider the universal relation –

R = {A, B, C, D, E, F, G, H, I, J} and the set of functional dependencies

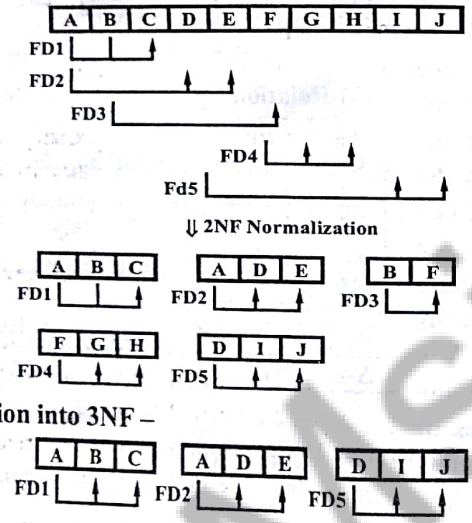
F = {A, B} → {C}, {A} → {D, E}, {B} → {F}, {F} → {G, H}, {D} → {I, J}

What is the key for R ? Decompose R into 2 NF, then 3 NF relations.

(R.G.P.V., Nov./Dec. 2007, Dec. 2014, June 2017)

Ans. The key of R is AB.

Normalization into 2NF –



Normalization into 3NF –

Q.31. Normalize the following database –

(Sno, Sname, Pno, Pname, Qty). ***(R.G.P.V., June 2009)***

Ans. **Step (i) [Reduction to 1NF]** – In the given schema, one Sno can contain duplicate parts. Obviously, this violates the principle of first normal form. Therefore, move all the repeating columns to another table. Thus, yielding the following two schemas

S (Sno, Sname) and SP (Sno, Pno, Pname, Qty)

Step (ii) [Reduction to 2NF] – Let us start with the S relation. This relation now contains two columns namely Sno, and Sname. The primary key is Sno. From this column, we can derive, the other nonkey columns, namely Sname. Therefore, this relation is in the second normal form.

Now, let us study the SP relation. This contains the columns Sno, Pno, Pname, Qty. We know that the primary key here is a composite key namely Sno + Pno. From this primary key, we can determine the Pname based on the Pno. We do not need the Sno for this purpose. Thus, all nonkey columns do not depend on the entire primary key, but some of them depend only on a part of the primary key. In other words, principle of 2NF is violated. Therefore, move columns that do not depend on the entire primary key to another table. Thus, yielding the following schemas

S(Sno, Sname), SP (Sno, Pno, Qty), and P(Pno, Pname)

Step (iii) [Reduction to 3NF] – The above obtained relations are in 3 NF.

Q.32. Normalize the following database –

empno, ename, job, sal, doj, mgr, deptno, dname, loc.

(R.G.P.V., June 2009, 2012)

Or

Normalize the following database (empno, ename, job, sal, day, mgr, depno, dname, loc). ***(R.G.P.V., Dec. 2010)***

Ans. **Step (i) [Reduction to 1NF]** – The given schema is decomposed into two relations to achieve 1NF.

emp (empno, ename, job, sal, doj, mgr), dept (deptno, dname, loc)

↓ primary key ↓ primary key

Step (ii) [Reduction to 2NF] – Here we decompose the relation into two relations

Emp (empno, ename, job, sal, doj, mgr, deptno)

↓ Foreign key

and, dept (deptno, dname, loc)

These relations do not have any partial dependencies. Each of the attributes is fully functionally dependent on the key attribute namely empno and deptno. Hence, these relations are in second normal form.

Step (iii) [Reduction to 3NF] – Both emp and dept relations are in 2NF. Also, each nonkey attribute in the table non-transitively depend on the entire primary key, empno and deptno, respectively.

DECOMPOSITION USING FD – DEPENDENCY PRESERVATION, LOSSLESS JOIN, BCNF, MULTI-VALUED DEPENDENCY, 4NF, JOIN DEPENDENCY AND 5NF

Q.33. Discuss the term decomposition.

Ans. The decomposition of a relation schema $R = \{A_1, A_2, \ldots\ldots, A_n\}$ is its replacement by a collection

$$\rho = \{R_1, R_2, \ldots\ldots, R_k\}$$ of subsets of R such that

$$R = R_1 \cup R_2 \cup \ldots\ldots \cup R_k$$

MSinVenter

There is no requirement that the R_iS be disjoint. One of the motivation of performing a decomposition is that it may eliminate some of problems. For example, in our pervious example of relation Sale (S#, Status, City, P#, Pname, Color, City). In this relation, we cannot insert the information that a supplier S1 lives in city Paris, if S1 does not currently supply any parts. It can be done by decomposing the relation Sale into relations S, SP, and P, as shown in table 4.4. This makes possible to insert the information that supplier S1 is located in city Paris, even though S1 does not currently supply any part, by simply inserting a tuple into relation S.

A decomposition may be lossy or lossless. A decomposition of a relation R into relations $R_1, R_2, R_3, ...,R_n$ is called a lossless-join decomposition (with respect to FDs F) if the relation R is always the natural join of the relations $R_1, R_2, ...,R_n$. It should be noted that natural join is the only way to recover the relation from the decomposed relations. There is no other set of operators that can recover the relation if the join cannot. If we are able to recover the original relation from natural join of decomposed relations, then this type of relation is lossless, otherwise relation is lossy.

Q.34. What is decomposition ? What are the desirable properties of decomposition ? ***(R.G.P.V., June 2014)***

Ans. **Decomposition** – Refer to Q.33.

Properties of decomposition are as follows –

(i) Non-loss or lossless decomposition

(ii) Dependency preservation

(iii) No need to repetition of information.

Q.35. Let R_1, R_2, R_3, ... R_n be the decomposition of schema U. Let u(U) be a relation and let $r_i = \pi R_i(u)$. Show that $u \subseteq r_1 \bowtie r_2 \bowtie r_3 ..... r_n$ ***(R.G.P.V., June 2013)***

Ans. Let us some tuple t in u.

Note that $r_i = \pi R_i(u)$ implies that $t[R_i] \in r_i (1 \le i \le n)$. So,

$$t[R_1] \bowtie t[R_2] \bowtie ..... \bowtie t[R_n] \in r_1 \bowtie r_2 \bowtie ..... \bowtie r_n$$

By definition of natural join,

$$t[R_1] \bowtie t[R_2] \bowtie ..... \bowtie t[R_n] = \pi_\alpha(\sigma_\beta(t[R_1] \times t[R_2] \times ..... \times t[R_n]))$$

Where the condition, β is satisfied if values of attributes with the same name in a tuple are equal and where $\alpha = U$. The cartesian product of single tuples generates on tuple. The selection process is satisfied because all attributes with the same name must have the same value since they are the projections from the same tuple. Finally, the projection clause removes duplicate attribute names.

As per the definition of decomposition, $U = R_1 \cup R_2 \cup ..... \cup R_n$.

It means that all attributes of t are in $t[R_1] \bowtie t[R_2] \bowtie ..... \bowtie t[R_n]$. That is,

t = Result of this join. Since t is any arbitrary tuple in u, so

$u \subseteq r_1 \bowtie r_2 \bowtie ..... \bowtie r_n$.

Q.36. Explain dependency preservation. ***(R.G.P.V., Nov./Dec. 2007)***

Or

Define the term dependency preservation. ***(R.G.P.V., Dec. 2011, 2013)***

Ans. It is desirable for a decomposition to have the lossless-join property, because it guarantees that any relation can be recovered from its projections. Another important property of a decomposition of relation schema R into $\rho = (R_1, ..., R_k)$ is that the set of dependencies F for R be implied by the projection of F onto the R_i's.

Also, if a relation schema has functional dependency, $A \rightarrow B$, then these two attributes are closely related. It would be useful to have such type of attributes in same relation. So it is desirable that, decompositions be such that, each dependency in F may be checked by looking at only one relation and that no joins need to be computed for checking dependencies.

For example, let us consider the relation R (A, B, C) with functional dependencies –

$$A \rightarrow C \quad \text{and} \quad B \rightarrow C$$

If we decompose this relation into two relations AB and BC, then we can easily check FD, $B \rightarrow C$ by looking at relation BC, but we would not be able to check dependency $A \rightarrow C$ by looking only at one relation.

So this decomposition is not desirable.

Consider another example of following table –

S#	*Status*	*City*
S1	10	Paris
S5	10	Holand

Relation S

This table having relation S(S#, City, Status) with functional dependencies–

$$S\# \rightarrow City$$

$$City \rightarrow Status$$

So, transitively we have

$$S\# \rightarrow Status$$

In fig. 4.6, dotted line shows transitive dependency, while bold lines show direct dependencies. We have a problem in this relation i.e., we cannot insert status of a city if that city does not have any supplier. This problem could be solved by decomposing relation S into two relations.

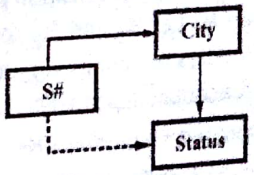


Fig. 4.6 FD's for Relation S

SC (S#, City) and CS (City, Status)

This decomposition can solve our problem, now we can easily insert the status of a city even if it does not have any supplier. Also, this decomposition is lossless.

At the same time, we can easily check dependency,

$$S\# \rightarrow City$$

and

$$City \rightarrow Status$$

By looking only at one relation at a time. If these two FD's hold, then transitive dependency, S# → Status, holds automatically. So, this decomposition preserves dependency. Hence, this decomposition is desirable.

Consider another possible decomposition of relation S –

SC (S#, City) and SS (S#, Status)

This decomposition is also lossless. But these two relations or projections are not independent i.e., here also we cannot insert the status of a city, if that city does not have any supplier.

Also, problem arises, if a supplier move from one city to another, then second relation also has to be updated for changing the status of that supplier. So we can say that, these projections are not independent.

Now, come to the point of dependency preservation,we can check FD's

$$S\# \rightarrow City \text{ and } S\# \rightarrow Status,$$

But, we cannot check City → Status by looking only at one relation. So, we can say that this decomposition does not preserve dependency and hence is not desirable.

Q.37. Explain lossless decomposition. ***(R.G.P.V., Nov./Dec. 2007)***

Or

Define the term lossless decomposition. ***(R.G.P.V., Dec. 2011, 2013)***

Ans. When a relation is decomposed into a number of smaller relations, then it is extremely important that the decomposition be lossless.

A decomposition $D = \{R_1, R_2, ..., R_m\}$ of R has the lossless join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following is true, where * denotes the natural join of all the relations in D –

$$*(\pi_{R1}(r), ..., \pi_{R_m}(r)) = r$$

Let R be a relation schema, and let F be a set of functional dependencies on R. Let R_1 and R_2 form a decomposition of R. This decomposition is a lossless (nonadditive) join decomposition of R if at least one of the following functional dependencies is in F^+ –

(i) $R_1 \cap R_2 \rightarrow R_1$ (ii) $R_1 \cap R_2 \rightarrow R_2$

In other words, if $R_1 \cap R_2$ forms a superkey of either R_1 or R_2, the decomposition of R is lossless-join decomposition.

For example, the relation schema lending_schema is as follows –

lending_schema = (branch_name, branch_city, assets, customer_name, loan_number, amount)

Now, we have to show that the decomposition of this schema is lossless-join decomposition. Suppose, lending_schema is decomposed into two schemas –

Branch_schema = (branch_name, branch_city, assets)

Loan_info_schema = (branch_name, customer_name, loan_number, amount)

Since branch_name → branch_city assets, the augmentation rule for functional dependency implies that

branch_name → branch_name branch_city assets

Since Branch_schema ∩ Loan_info_schema = {branch_name}, it follows that our initial decomposition is a lossless-join decomposition.

Next, we decompose Loan_info_schema into

Loan_schema = (loan_number, branch_name, amount)

Borrower_schema = (customer_name, loan_number). This step results in a lossless-join decomposition since loan_number is a common attribute and loan_number → amount branch_name.

Q.38. What do you mean by decomposition of a relation ? Consider the relational scheme R(A, B, C, D, E, F) and FDs

$$A \rightarrow BC,\ C \rightarrow A,\ D \rightarrow E,\ F \rightarrow A,\ E \rightarrow D$$

Is the decomposition of R into R_1(A, C, D), R_2(B, C, D) and R_3(E, F, D) lossless ? Explain the requirements for lossless decomposition and dependency preserving. ***(R.G.P.V., Dec. 2011)***

Ans. **Decomposition of a Relation** – Refer to Q.33.

Step (i) – Since the original relation has 6 attributes and the original relation has been decomposed into 3 relations, we need to create a table with 6 columns and 3 rows. The column of the table are named A, B, C, D, E and F respectively. The rows of the table are named R_1, R_2 and R_3. For explanation purposes we have renamed the attributes A_1,...,A_6. The table is shown below –

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$	$F(A_6)$
R_1						
R_2						
R_3						

Step (ii) – The attributes of the scheme of R_1 are – $A(A_1)$, $C(A_3)$ and $D(A_4)$. Therefore, we place a_1, a_3 and a_4 under these columns, respectively. The remaining entries of this row are filled with b_{12}, b_{15} and b_{16}.

The attributes of the scheme R_2 are – $B(A_2)$, $C(A_3)$ and $D(A_4)$. Therefore, we place a_2, a_3 and a_4 under these columns respectively. The remaining entries of this row are filled with b_{21}, b_{25} and b_{26}.

The attributes of the scheme R_3 are – $E(A_5)$, $F(A_6)$ and $D(A_4)$. Therefore, we place a_5, a_6 and a_4 under these columns respectively. The remaining entries of this row are filled with b_{31}, b_{32} and b_{33}.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$	$F(A_6)$
R_1	a_1	b_{12}	a_3	a_4	b_{15}	b_{16}
R_2	b_{21}	a_2	a_3	a_4	b_{25}	b_{26}
R_3	b_{31}	b_{32}	b_{33}	a_4	a_5	a_6

Step (iii) Ist Time – Assuming $A \rightarrow BC$ we check if the rows of table have the same value under the columns that make up the determinant of the FD. Since the rows do not have identical values, we repeat step (iii) and consider another FD.

IInd Time – Assuming $C \rightarrow A$ we check if the rows of the table have the same value under the columns that make up the determinant of the FD. Since the rows do not have identical value, we repeat step (iii) and consider another FD.

IIIrd Time – Assuming $D \rightarrow E$ we check if the rows of the table have the same value under the columns that make up the determinant of the FD. Rows has value a_4 under column D. Therefore, we need to equate all the corresponding entries for these rows under column E. Since the entry under the column E is a_5 (for row R_3), we make all the remaining b_{ij}'s equal to a_5 in this column.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$	$F(A_6)$
R_1	a_1	b_{12}	a_3	a_4	a_5	b_{16}
R_2	b_{21}	a_2	a_3	a_4	a_5	b_{26}
R_3	b_{31}	b_{32}	b_{33}	a_4	a_5	a_6

IV Time – Assuming $F \rightarrow A$ we check if the rows of the table have the same value under the columns that make the determinant of the FD. Since the rows do not have identical value, we repeat step (iii) and consider another FD.

V Time – Assuming $E \rightarrow D$ we check if the rows of the table have the same value under the columns that make the determinant of the FD. Rows has value a_5 under the column E. Therefore, we need to equate all the corresponding entries for these rows under the column D, which is already exist.

VIth Time – There are no more FDs to consider and the table cannot undergo any more changes. Therefore, we move to step (iv).

Step (iv) – We now look for a row that has all a_i's. Since there is no row in the table that has all a_i's in its entries the decomposition is lossy.

Whenever a relation is decomposed it is necessary to ensure that the data in the original relation is represented faithfully by the data in the relations that are the result of the decomposition, i.e., we need to make sure that we can recover the original relation from the new relations that have replaced it.

Dependency preserving property requires that the decomposition satisfy all the FDs that are satisfied by the original relation. The reason this is desirable is that the set of functional dependencies that are satisfied by a relation define integrity constraints that the relation needs to meet.

Q.39. Suppose that we decompose the schema $R = \{P, Q, R, S, T\}$ into $R_1 = \{P, Q, R\}$ and $R_2 = \{P, S, T\}$. Verify whether this decomposition is lossless or lossy if the functional dependencies $P \rightarrow QR$, $RS \rightarrow T$, $Q \rightarrow S$ and $T \rightarrow P$ holds. Also verify whether the decomposition is dependency preserving. ***(R.G.P.V., Dec. 2017)***

Ans. **Step (i)** – Since the original relation has 5 attributes and the original relation has been decomposed into 2 relations, we need to create a table with 5 columns and 2 rows. The columns of the table are named P, Q, R, S and T respectively. The rows of the table are named R_1 and R_2. For explanation purposes we have renamed the attributes A_1, ... A_5. The table is shown below –

	$P(A_1)$	$Q(A_2)$	$R(A_3)$	$S(A_4)$	$T(A_5)$
R_1					
R_2					

Step (ii) – The attributes of the scheme of R_1 are – $P(A_1)$, $Q(A_2)$, $R(A_3)$. Therefore, we place a_1, a_2 and a_3 under these columns, respectively. The remaining entries of this row are filled with b_{14} and b_{15}.

The attributes of the scheme of R_2 are $P(A_1)$, $S(A_4)$, $T(A_5)$. Therefore, we place a_1, a_4 and a_5 under these columns respectively. The remaining entries of this row are filled with b_{22} and b_{23}.

	$P(A_1)$	$Q(A_2)$	$R(A_3)$	$S(A_4)$	$T(A_5)$
R_1	a_1	a_2	a_3	b_{14}	b_{15}
R_2	a_1	b_{22}	b_{23}	a_4	a_5

Step (iii) Ist Time – Assuming $P \rightarrow QR$ we check if the two rows of table have the same value under the columns that make up the determinant of the FD. Rows R_1 and R_2 has values a_1 under column P. Therefore, we need to equate all the corresponding entries for these rows under columns Q and R. Since entry under column Q is a_2 (for row R_1), we make all the remaining b_{ij}'s equal to a_2 in this column. The entry under R column is a_3, we make the corresponding entry a_3 for row R_3.

	$P(A_1)$	$Q(A_2)$	$R(A_3)$	$S(A_4)$	$T(A_5)$
R_1	a_1	a_2	a_3	b_{14}	b_{15}
R_2	a_1	a_2	a_3	a_4	a_5

IInd Time – Assuming RS → T, we check for rows that have the same value in the columns R and S. In this case, rows do not have identical values we repeat step (iii) and consider another FD.

IIIrd Time – Assuming Q → S, we check if the two rows of the table have the same value under the columns that make up the determinant of the FD. Since the rows have identical values under column A_2, we equate the corresponding entries under A_4 column. The entries under A_4 are b_{14} and a_4. so we take entry a_4 under A_4 column arbitrarily.

	$P(A_1)$	$Q(A_2)$	$R(A_3)$	$S(A_4)$	$T(A_5)$
R_1	a_1	a_2	a_3	a_4	b_{15}
R_2	a_1	a_2	a_3	a_4	Q_5

IVth Time – Assuming T → P, we check if the two rows of the table have the same value under the columns that make up the determinant of the FD. Since the rows do not have identical values, we repeat step (iii) and consider another FD.

Vth Time – There are no more FDs to consider and the table cannot undergo any more changes. Therefore we move to step (iv).

Step (iv) – We now look for a row that has all a_i's. Since row R_2 has become $a_1\ a_2\ a_3\ a_4\ a_5$, the decomposition is lossless. Hence, this decomposition is dependency preserving.

Q.40. Suppose that we decompose the scheme R = (A, B, C, D, E) into (A, B, C) and (A, D, E). Show that this decomposition is a lossless join decomposition if the following function dependency hold – A → BC, CD → E, B → D. ***(R.G.P.V., May 2018)***

Sol. **Step (i)** – Since the original relation has 5 attributes and the original relation has been decomposed into 2 relations, we need to create a table with 5 columns and 2 rows. The columns of the table are named A, B, C, D and E respectively. The rows of the table are named R_1 and R_2. For explanation purposes we have renamed the attributes A_1,...., A_5. The table is shown in side.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$
R_1					
R_2					

Step (ii) – The attributes of the scheme of R_1 are – $A(A_1)$, $B(A_2)$, $C(A_3)$. Therefore, we place a_1, a_2 and a_3 under these columns, respectively. The remaining entries of this row are filled with b_{14} and b_{15}.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$
R_1	a_1	a_2	a_3	b_{14}	b_{15}
R_2	a_1	b_{22}	b_{23}	a_4	a_5

The attributes of the scheme of R_2 are – $A(A_1)$, $D(A_4)$, $E(A_5)$. Therefore, we place a_1, a_4 and a_5 under these columns respectively. The remaining entries of this row are filled with b_{22} and b_{23}.

Step (iii) Ist Time – Considering A → BC we check if the two rows of table have the same value under the columns that make up the determinant of the FD. Rows R_1 and R_2 has values a_1 under column A. Therefore, we need to equate all the corresponding entries for these rows under columns B and C. Since entry under column B is a_2 (for row R_1), we make all the remaining b_{ij}'s equal to a_2 in this column. The entry under C column is a_3, we make the corresponding entry a_3 for this column.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$
R_1	a_1	a_2	a_3	b_{14}	b_{15}
R_2	a_1	a_2	a_3	a_4	a_5

IInd Time – Considering CD → E, we check for rows that have the same value in the columns C and D. In this case, rows do not have identical values we repeat step (iii) and consider another FD.

IIIrd Time – Considering B → D, we check if the two rows of the table have the same value under the columns that makeup the determinant of the FD. Since the rows have identical values under column A_2, we equate the corresponding entries under A_4 column. The entries under A_4 are b_{14} and a_4, so we take entry a_4 under A_4 column arbitrarily.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$
R_1	a_1	a_2	a_3	a_4	b_{15}
R_2	a_1	a_2	a_3	a_4	a_5

IVth Time – There are no more FDs to consider and the table cannot undergo any more changes. Therefore we move to step (iv).

Step (iv) – We now look for a row that has all a_i's. Since row R_2 has become $a_1a_2a_3a_4a_5$, the decomposition is lossless.

Q.41. Given the relation r (X, Y, W, Z , P, Q) and the set F = {XY → W, XW → P, PQ → Z, XY → Q}. Consider the decomposition R_1(Z, P, Q), R_2 (X, Y, Z, P, Q). In this decomposition lossless or lossy ? Use the lossless – join algorithm. ***(R.G.P.V., Dec. 2013)***

Ans. **Step (i)** – Create a table having six columns (one column corresponds to one attribute) and two rows (one row corresponds to one relation of the decomposition). Name each column with the name of one attribute and each row with the name of a relation.

	$X(A_1)$	$Y(A_2)$	$W(A_3)$	$Z(A_4)$	$P(A_5)$	$Q(A_6)$
R_1						
R_2						

MRP Work

Step (ii) – Fill in the entries as follows –

The attributes of the scheme of R_1 are – Z (A_4), P (A_5), Q (A_6). Hence, we put a_4, a_5 and a_6 under these columns, respectively. The rest of the entries of this row are filled with b_{11}, b_{12}, and b_{13} respectively.

The attributes of the scheme of R_2 are X (A_1), Y (A_2), Z (A_4), P (A_5), Q(A_6). Hence, we put a_1, a_2, a_4, a_5, and a_6 under these columns respectively. The rest of the entry of this row is filled with b_{23}.

	$X(A_1)$	$Y(A_2)$	$W(A_3)$	$Z(A_4)$	$P(A_5)$	$Q(A_6)$
R_1	b_{11}	b_{12}	b_{13}	a_4	a_5	a_6
R_2	a_1	a_2	b_{23}	a_4	a_5	a_6

Step (iii) (Ist Time) – Let us XY → W. Here, we check for the two rows of the table containing identical value under the columns that form the determinant of the FD. Because the rows do not have same values, we repeat step (iii) and consider another FD.

Step (iii) (IInd Time) – Let us XW → P. Here, we check for the two rows of the table having identical values under the columns that form the determinant of the FD. Because the rows do not have same values, we repeat step (iii) and consider another FD.

Step (iii) (IIIrd Time) – Let us PQ → Z. Here, we check for the two rows of the table having identical values under the columns that form the determinant of the FD. Because, the rows contain identical values under the attributes P and Q respectively, we make equal the entries of the two rows under attribute Z. Because all the values under Z are already equal no changes are required we repeat step (iii) and consider another FD.

Step (iii) (IVth Time) – Let us XY → Q. Here, we check for the two rows of the table having identical values under the columns that form the determinant of the FD. Because the rows do not have same values, the table will remain unchanged. There are no more FDS to consider, so we move to step (iv).

Step (iv) (Ist Time) – So, there are no rows contains all a_i's, so the decomposition is lossy.

Q.42. Discuss the purpose of Boyce codd normal form and describe how BCNF differ from and is stronger than 3 NF. Illustrate your answer with an example. ***(R.G.P.V., June 2011)***

Or

Define BCNF. How does it differ from 3NF ? What is it considered a stronger form of 3NF ? Explain with neat diagram. (R.G.P.V., June 2017)

Or

Write short note on BCNF. ***(R.G.P.V., Dec. 2017)***

Ans. Boyce-codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF, because every relation in BCNF is also in 3NF. However, a relation in 3NF is not necessarily in BCNF.

The BCNF can be defined as follows –

A relation is in BCNF, if and only if every nontrivial, left irreducible FD has a candidate key as its determinant.

Or

A relation is in BCNF, if and only if the only determinants are candidate keys.

Or

A relation R is said to be in BCNF, if whenever X → A holds in R and A is not in X, then X is a candidate key for R.

In other words, the only arrows in the FD diagram are arrows out of candidate keys, and there are no others, meaning there are no arrows that can be eliminated by the normalization procedure.

Relations Sale and S which are not in 3NF, are not in BCNF, while the relations SP, SC, and CS which are in 3NF, are also in BCNF. Because in relation Sale, we have four determinants {S#}, {P#}, {CITY}, {S#, P#}, of these only {S#, P#} is a candidate key. So Sale is not in BCNF. Similarly, in relation S we have 2 determinants {S#}, {CITY}, and CITY is not a candidate key. So S is also not in BCNF.

While relation SP, SC, and CS are each in BCNF, because in each case the Sale candidate key is the only determinant in the relation.

Mostly, relations that are in 3NF are also in BCNF. Infrequently, a 3NF relation is not in BCNF and this happens only if –

(i) The candidate keys in the relation are composite keys.

(ii) There is more than one candidate keys in the relation, and

(iii) The keys are not disjoint, that is, some attributes in the keys are common.

Now, we take another example that is in BCNF.

Table 4.6 Relation Player

Name	*Language*	*Hobby*
Nitin	Hindi	Football
Nitin	English	Swimming
Nitin	Hindi	Swimming
Nitin	English	Football
Jitin	French	Jogging
Jitin	Spanish	Jogging

Table 4.7

Name	*Language*	*Hobby*
Nitin	Hindi	Football
Nitin	English	Football
Nitin	Hindi	Swimming
Nitin	English	Swimming
Jitin	French	Jogging
Jitin	Spanish	Jogging
Jitin	French	Singing

Table 4.8

Name	*Language*	*Hobby*
Nitin	Hindi	Football
Nitin	English	Football
Nitin	Hindi	Swimming
Nitin	English	Swimming
Jitin	French	Jogging
Jitin	Spanish	Jogging
Jitin	French	Singing
Jitin	Spanish	Singing

Here, we are supposing name as a unique key. Here, a player can know more than one languages and may have more than one hobbies too. So, each of the attributes is a primary key. So it is in BCNF. But this also suffers from some anomalies, such as, if we want to add one more hobby of Jitin that is singing. Then we can do in following manner –

But this would mean that Jitin has hobby singing in French language but he does not know singing in Spanish language. So, adding a tuple with hobby singing and language Spanish, would result

BCNF faces some anomalies –

(i) INSERT – If we want to add a single tuple then several others have to be added as given above.

(ii) DELETE – Similar to insertion, if we delete a single tuple, several other tuples also have to be deleted.

For example, if we want to delete (Nitin, Hindi, Football) then (Nitin, English, Football) also has to be deleted.

Q.43. What do you mean by normalization ? Explain BCNF and 3NF with suitable example. ***(R.G.P.V., Dec. 2011)***

Ans. **Normalization** – Refer to Q.15.

BCNF – Refer to Q.42.

3NF – Refer to Q.25 and Q.27.

Q.44. List the three design goals for relational database, and explain why each is desirable. ***(R.G.P.V., Dec. 2009)***

Or

Discuss the important design goals for relational database, and explain why each is desirable. ***(R.G.P.V., Dec. 2016)***

Ans. The three design goals for relational database are as follows –

(i) BCNF

(ii) Lossless-join decomposition

(iii) Dependency preservation.

If we cannot achieve all three, we accept

(a) 3NF

(b) Lossless join decomposition

(c) Dependency preservation.

BCNF – Refer to Q.42.

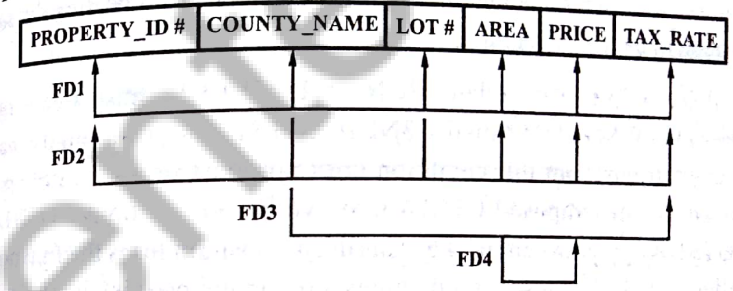
3NF – Refer to Q.25 and Q.27.

Dependency Preservation – Refer to Q.36.

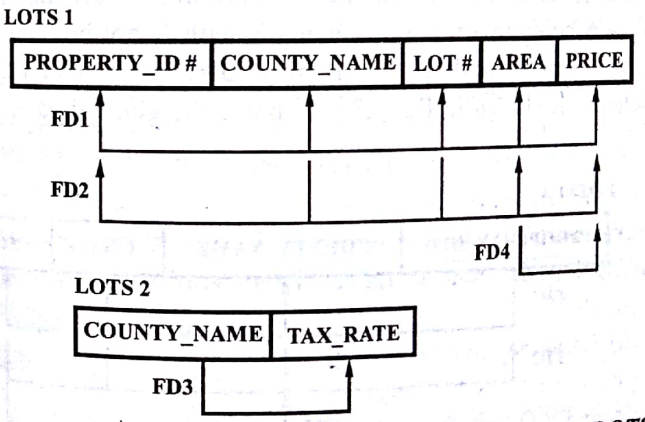
Lossless-join Decomposition – Refer to Q.37.

Q.45. Show that if relational schema is in BCNF it is in 3NF but the reverse is not true. ***(R.G.P.V., June 2007)***

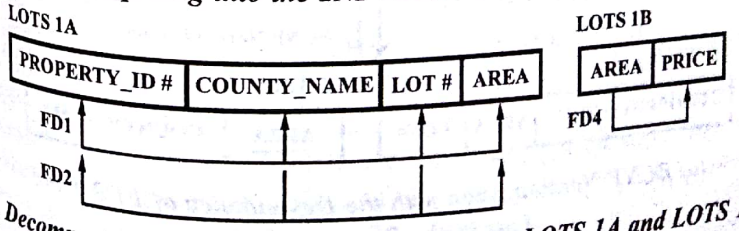
Ans. Fig. 4.8 shows lots relation schema. Fig. 4.7 (a) contains four functional dependencies, FD1 through FD4. Assume that, there are lots from only two counties – Dekalb and Fulton. Assume also that lot sizes for in Dekalb county are .5, .6, .7, .8, .9 and 1.0 acres, whereas lot sizes in fulton county are 1.1, 1.2,.., 1.9 and 2.0 acres. In this situation, we will have the additional functional dependency FD5 : area → county_name. If we add this to other dependencies, the relation LOTS1A still is in 3NF because county_name is a prime attribute.



(a) The LOTS Relation Schema and its Functional Dependencies FD1 Through FD4

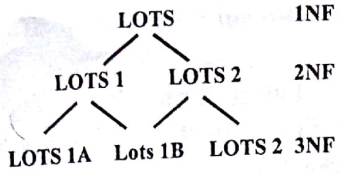


(b) Decomposing into the 2NF Relations LOTS 1 and LOTS 2



(c) Decomposing LOTS 1 into the 3NF relations LOTS 1A and LOTS 1B

Mahaveer



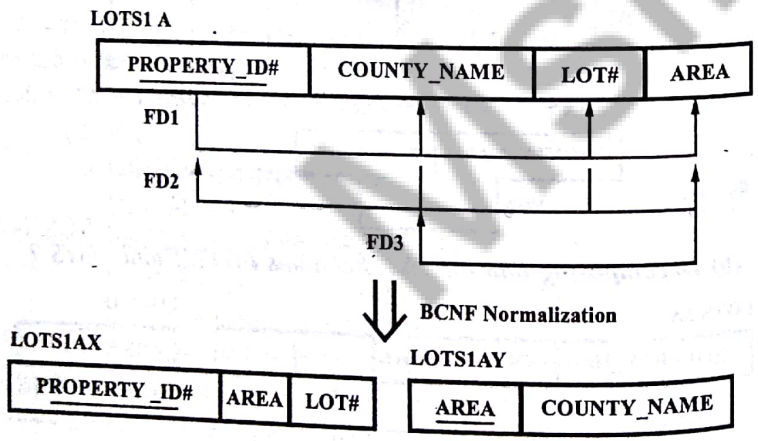
(d) Summary of Normalization of LOTS

Fig. 4.7 Normalization to 2NF and 3NF

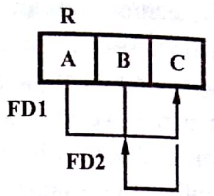
A lot of area that determines the county can be represented by 16 tuples in a separate relation R(area, county_name), since there are only 16 possible area values. This representation reduces the redundancy of repeating the same information in the LOTS1A tuples.

In this example, FD5 violates BCNF in LOTS1A because area is not a superkey of LOTS1A.FD5 satisfies 3NF in LOTS1A because county_name is a prime attribute, but this condition does not exist in the definition of BCNF. We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1 AY as shown in fig. 4.8. This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation.

In practice, some relation schemas that are in 3NF are also in BCNF. Only if $X \rightarrow A$ holds in a relation schema R with X not being a superkey and A being a prime attribute will R be in 3NF but not in BCNF. The relation schema R shown in fig. 4.8 (b) shows the general case of such a relation.



(a) BCNF Normalization with the Dependency of FD2 being Lost in the Decomposition



(b) A Relation R in 3NF but not in BCNF

Fig. 4.8 Boyce-codd Normal Form

Table 4.9 shows a relation TEACH with the following dependencies –
FD1 : {STUDENT, COURSE} → INSTRUCTOR
FD2 : INSTRUCTOR → COURSE

Table 4.9 A Relation TEACH that is in 3NF but not in BCNF

TEACH

STUDENT	*COURSE*	*INSTRUCTOR*
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating System	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

In this figure {STUDENT, COURSE} is a candidate key for this relation and the dependencies follow the pattern in fig. 4.8 (b). Hence this relation is in 3NF but not in BCNF.

Q.46. What is BCNF ? Give an example of a relation schema R and set of dependencies such that R is in BCNF, but not in 3NF. ***(R.G.P.V., Dec. 2016)***

Ans. **BCNF** – Refer to Q.42.

Example – Refer to Q.45.

Q.47. Explain the similarities and dissimilarities between BCNF and 3rd normal form. ***(R.G.P.V., June 2009, Dec. 2010)***

Or

Compare BCNF versus 3NF. ***(R.G.P.V., Dec. 2012)***

Ans. 3NF and BCNF are the two normal forms for relational database schemas, there are advantages to 3NF in that we know that it is always possible to obtain a 3NF design without sacrificing a lossless join or dependency preservation. Disadvantages of 3NF is that, if we do not eliminate all transitive relations schema dependencies, we may have to use null values to represent

some of the possible meaningful relationships among data items and there is the problem of repetition of information. As an illustration of the null value problem, consider again the banker_schema and its associated functional dependencies. Since banker_name → branch_name, we may want to represent relationships between values for banker_name and values for branch_name in our database. If we are to do so, however, either there must be a corresponding value for customer_name, or we must use a null value for the attribute of customer_name.

As an illustration of the repetition of information problem, consider the instance of banker_schema in table 4.10. Notice that the information indicating that Johnson is working at the perryridge branch is repeated our goals of database design with functional dependencies are – BCNF, Lossless Join, Dependency preservation. Since it is not always possible to satisfy all three, we may be forced to choose between BCNF and dependency preservation with 3NF.

Table 4.10 An Instance of banker_schema

customer_name	*banker_name*	*branch_name*
Jones	Johnson	Perryridge
Smith	Johnson	Perryridge
Hayes	Johnson	Perryridge
Jackson	Johnson	Perryridge
Curry	Johnson	Perryridge
Turner	Johnson	Perryridge

It is worth noting that SQL does not provide a way of specifying functional dependencies, except for the special case of declaring superkeys by using the ***primary key*** or ***unique*** constraints. It is possible, although a little complicated, to write assertions that enforce a functional dependency, unfortunately, testing the assertions would be very expensive in most database systems. Thus even if we had a dependency-preserving decomposition, if we use standard SQL we would not be able to efficiently test a functional dependency whose left-hand side is not a key.

Given a BCNF decomposition that is not dependency preserving, we consider each dependency in a minimum cover Fc that is not preserved in the decomposition. For each such dependency $\alpha \rightarrow \beta$, we define a materialized view that computes a join of all relations in the decomposition and projects the result on $\alpha\beta$. The functional dependency can be easily tested on the materialized view, by means of a constraint *unique* (α). On the negative side, there is a space and time overhead due to the materialized view, but on the positive side, the application programmer need not worry about writing code to keep redundant data consistent on updates, it is the job of the database system to maintain the materialized view, that is keep up to data when the database is updated. Thus, in case we are not able to get a dependency-preserving BCNF decomposition, it is generally preferable to opt for BCNF and use techniques such as materialized views to reduce the cost of checking functional dependencies.

Q.48. Explain multivalued dependency. ***(R.G.P.V., Nov./Dec. 2007)***

Or

What is a multivalued dependency ? What type of constraint does it specify ? When does it arise ? ***(R.G.P.V., June 2010)***

Or

Define multivalued dependency. What do you understand by trivial multivalued dependency ? ***(R.G.P.V., Dec. 2015)***

Ans. Functional dependencies rule out certain tuples from being in a relation i.e., if $A \rightarrow B$ then we cannot have two tuples with the same A value but different B values. On the other hand, multivalued dependencies do not rule out the existence of these tuples. That is why, functional dependencies sometimes referred to as ***equality-generating dependencies*** and multivalued dependencies are referred to as ***tuple-generating dependencies.***

Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The multivalued dependency $\alpha \rightarrow\rightarrow \beta$ holds on R if, in any legal relation r(R), for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

Whenever $\alpha \rightarrow\rightarrow \beta$ holds we say that α multidetermines y.

Fig. 4.9 shows a tabular representation of t_1, t_2, t_3 and t_4. Intuitively, the multivalued dependency $\alpha \rightarrow\rightarrow \beta$ say that the relationship between α and β is independent of the relationship between α and $R - \beta$. If the multivalued dependency $\alpha \rightarrow\rightarrow \beta$ is satisfied by all relations on schema R, then $\alpha \rightarrow\rightarrow \beta$ is a trivial multivalued dependency on schema R. So, $\alpha \rightarrow\rightarrow \beta$ is trivial if $\beta \subseteq \alpha$ or $\beta \cup \alpha = R$.

	α	β	$R-\alpha-\beta$
t_1	$a_1 \cdots a_i$	$a_{i+1} \cdots a_j$	$a_{j+1} \cdots a_n$
t_2	$a_1 \cdots a_i$	$b_{i+1} \cdots b_j$	$b_{j+1} \cdots b_n$
t_3	$a_1 \cdots a_i$	$a_{i+1} \cdots a_j$	$b_{j+1} \cdots b_n$
t_4	$a_1 \cdots a_i$	$b_{i+1} \cdots b_j$	$a_{j+1} \cdots a_n$

Fig. 4.9 Tabular Representation of $\alpha \rightarrow\rightarrow \beta$

Q.49. Define and explain 4NF with an example.

Or

Prove that a relation which is in 4NF must be in BCNF. ***(R.G.P.V., June 2008, Dec. 2008)***

Or

Explain the cases where 4th normal form is applicable. How does it remove the anomalies ? ***(R.G.P.V., June 2012)***

Uniquer

Ans. A relation R is in 4NF if and only if, whenever there exists subsets A and B of the attributes of R such that the nontrivial multivalued dependency A $\rightarrow\rightarrow$ B is satisfied then all attributes of R also functionally dependent on A.

Or

A relation is in 4NF if it is in BCNF and it has no multivalued dependency.

The EMP relation of fig. 4.10 (a) is not in 4NF because in the nontrivial MVDs ENAME $\rightarrow\rightarrow$ PNAME and ENAME $\rightarrow\rightarrow$ DNAME, ENAME is not a superkey of EMP. We decompose EMP into EMP-PROJECTS and EMP-DEPENDENTS as shown in fig. 4.10 (b). Now, both EMP-PROJECTS and EMP-DEPENDENTS are in 4NF, because the MVDs ENAME $\rightarrow\rightarrow$ PNAME in EMP-PROJECTS and ENAME $\rightarrow\rightarrow$ DNAME in EMP-DEPENDENTS are trivial MVDs.

EMP

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(a) The EMP Relation with Two MVDs ENAME $\rightarrow\rightarrow$ PNAME and ENAME $\rightarrow\rightarrow$ DNAME

EMP-PROJECTS

ENAME	PNAME
Smith	X
Smith	Y

EMP-DEPENDENTS

ENAME	DNAME
Smith	John
Smith	Anna

(b) Decomposing EMP into Two Relations in 4NF

Fig. 4.10

Every 4NF schema is in BCNF. To see this fact, if a schema R is not in BCNF then there is a nontrivial functional dependency $\alpha \rightarrow \beta$ holding on R where α is not a super key. Since $\alpha \rightarrow \beta$ implies $\alpha \rightarrow\rightarrow \beta$, R cannot be in 4NF.

Q.50. Describe about the multivalued dependencies and fourth normal form with suitable example. ***(R.G.P.V., June 2015)***

Or

Explain multi-valued dependencies and also describe 4NF with appropriate example. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.48 and Q.49.

Q.51. Give an example of a relation schema R and a set of dependencies such that R is in BCNF, but not in 4NF. (R.G.P.V., Dec. 2009, June 2015)

Ans. Assume that we have a relation with attributes course, teacher, and book, which we denote as CTB (see table 4.11). The meaning of a tuple is that teacher T can teach course C, and book B is a recommended text for the course.

Table 4.11 BCNF Relation with Redundancy that is Revealed by MVDs

course	teacher	book
Physics101	Green	Mechanics
Physics101	Green	Optics
Physics101	Brown	Mechanics
Physics101	Brown	Optics
Math301	Green	Mechanics
Math301	Green	Vectors
Math301	Green	Geometry

CTB satisfies no functional dependencies at all apart from trivial ones such as course $\rightarrow$ course. In fact, CTB is in BCNF, since it is all key relation.

The relation CTB is not in 4NF because C $\rightarrow\rightarrow$ T is a nontrivial MVD and C is not a key. We can eliminate the resulting redundancy by decomposing CTB into CT and CB. Each of there relations is then in 4NF.

Q.52. Explain the join dependency and fifth normal form (PJ/NF).

Or

What is meant by projection join dependency ? (R.G.P.V., Dec. 2010)

Or

What is meant by projection dependency ? (R.G.P.V., June 2012)

Or

Discuss join dependencies on fifth normal form and explain why 5 NF ? (R.G.P.V., Dec. 2014)

Ans. There may be no functional dependency in a relation schema R that violates any normal form up to BCNF and there may be no nontrivial MVD in R that violates 4NF. We then use another dependency called the join dependency. It carries out a multiway decomposition into fifth normal form (5NF).

A join dependency (JD), denoted by JD(R_1, R_2, ..., R_n), specified on relation schema R, specifies a constraint on the states r of R. The constraint states that every legal state r of R should have a lossless join decomposition into R_1, R_2, ..., R_n i.e., for every such r we have

$$(\Pi_{R1}, (r), \Pi_{R2}(r), ..., \Pi_{Rn}(r)) = r$$

An MVD is a special case of a JD where n = 2. That is, a JD denoted as JD (R_1, R_2) implies an MVD ($R_1 \cap R_2$) $\rightarrow\rightarrow$ ($R_1 - R_2$). A join dependency JD (R_1, R_2, ..., R_n), specified on relation schema R, is a trivial JD if one of the relation schemas R_i in JD (R_1, R_2, ..., R_n) is equal to R. Such a dependency

is called trivial because it has the lossless join property for any relation state r of R and hence does not specify any constraint on R.

A relational schema R is in fifth normal form (5NF) or projection join normal form (PJNF) with respect to a set F of functional multivalued, and join dependencies if, for every nontrivial join dependency JD (R_1, R_2, ..., R_n) in F+ (that is implied by F), every R_i is a superkey of R.

For an example of JD, consider the SUPPLY all-key relation of fig. 4.11(a). Suppose that the following constraint always holds – Whenever a supplier s supplies part p, and a project j uses part p, and the supplier s supplies at least one part to project j, then supplier s will also be supplying part p to project j. This constraint can be restated as specifying a join dependency JD(R_1, R_2, R_3) among the three projections R1 (SNAME, PARTNAME), R_2 (SNAME, PROJNAME), and R_3 (PARTNAME, PROJNAME) of SUPPLY. If this constraint holds, the tuples below the dotted line in fig. 4.11(a) must exist in any legal state of the SUPPLY relation that also contains the tuples above the dotted line. Fig. 4.11(b) shows how the SUPPLY relation with the join dependency is decomposed into three relations R_1, R_2 and R_3 that are each in 5 NF.

SUPPLY

SNAME	PARTNAME	PROJNAME
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(a) The Relation SUPPLY with no MVDs Satisfies 4NF but does not Satisfy 5NF if the JD(R_1, R_2, R_3) Holds

R_1

SNAME	PARTNAME
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

SNAME	PROJNAME
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

PARTNAME	PROJNAME
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

(b) Decomposing SUPPLY into three 5NF Relations

Fig. 4.11

Q.53. Explain BCNF and 3NF and 5NF with suitable example. ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.42, Q.25, Q.27 and Q.52.

Q.54. Describe the term MVD in the context of relational database management system by giving an example. Discuss 4NF and 5NF also. ***(R.G.P.V., Dec. 2011)***

Ans. **MVD** – Refer to Q.48.

4NF – Refer to Q.49.

5NF – Refer to Q.52.

Q.55. Explain the different levels of normalization with a suitable example. ***(R.G.P.V., June 2010)***

Or

What are basic normalization techniques ? Explain each with suitable database. ***(R.G.P.V., June 2009)***

Or

What is normalization ? Explain 1NF, 2NF, 3NF and BCNF with simple example. ***(R.G.P.V., June 2010)***

Or

What is normalization ? Give the various normal forms of relational schema and define a relation which is in BCNF and explain with suitable example. ***(R.G.P.V., Dec. 2012)***

Or

Define normalization. What is the importance of normalization in database design ? Explain 1 NF, 3 NF and BCNF with example. ***(R.G.P.V., June 2014)***

Or

What is normalization ? Discuss various normal forms with the help of examples. ***(R.G.P.V., Dec. 2013)***

Ans. **Normalization** – Refer to Q.15.

1 NF – Refer to Q.18 and Q.19.

2 NF – Refer to Q.21 and Q.23.

3 NF – Refer to Q.25 and Q.27.

BCNF – Refer to Q.42.

4 NF – Refer to Q.49.

5 NF – Refer to Q.52.

MsInventer

Q.56. Consider the relation schema R(P, Q, R, S) and the functional dependencies PQ → R, R → S and S → P holds on R. List all the keys of relation R. ***(R.G.P.V., Dec. 2017)***

Ans. Given,

relation schema R(P, Q, R, S)

functional dependencies {PQ → R, R → S, S → P}

A relation R with four attributes P, Q, R, S having given dependencies contains the following keys – PQ.

⌘⌘

UNIT 5

INTRODUCTION OF TRANSACTION, TRANSACTION PROCESSING AND RECOVERY

Q.1. Write short note on transaction processing. ***(R.G.P.V., June 2007)***

Or

What do you mean by a transaction ? ***(R.G.P.V., June 2015)***

Ans. A transaction is a logical unit of database processing that includes one or more database access operations such as insertion, deletion, modification, and retrieval. The database operations that form a transaction can either be embedded within an application program or they can be specified interactively through a high-level query language such as SQL, where it is delimited by statements of the form ***begin transaction*** and ***end transaction.*** In this case, all database access operations between the two are considered as forming one transaction. A single application program may contain more than one transaction if it contains several transaction boundaries.

Transactions access data using two operations –

(i) Read (X) – It transfers the data item X from the database to a local buffer belonging to the transaction that executed the read operation.

(ii) Write (X) – It transfers the data item X from the local buffer of the transaction that executed the write back to the database.

Fig. 5.1 shows examples of two very simple transactions. The read-set of a transaction is the set of all items that the transaction reads, and the write-set is the set of all items that the transaction writes. For example, the read-set of T_1 in fig. 5.1 is {X, Y} and its write-set is also {X, Y}.

T_1
read (X);
X := X – N;
write (X);
read (Y);
Y := Y + N;
write (Y);

(a) Transaction T_1

T_2
read (X);
X := X + M
write (X);

(b) Transaction T_2

Fig. 5.1 Two Sample Transactions

MsinVenr

Q.2. Explain four important properties of transaction that a DBMS must ensure to maintain database. ***(R.G.P.V., June 2011)***

Or

What are the properties of transaction ? ***(R.G.P.V., Dec. 2015)***

Or

What are the ACID properties ? ***(R.G.P.V., Dec. 2014)***

Or

Write the basic properties of a transaction. ***(R.G.P.V., June 2016)***

Or

Database system implementers have paid much more attention to the ACID properties than have file system implementers. Why might this be the case ? ***(R.G.P.V., Dec. 2009)***

Ans. The database system maintains the following properties of the transactions to ensure integrity of the data –

(i) Atomicity – A transaction is an atomic unit of processing. It is either performed in its entirety or not performed at all.

(ii) Consistency – Transactions preserve database consistency. That is, a transaction transforms a consistent state of the database into another consistent state, without necessarily preserving consistency at all intermediate points.

(iii) Isolation – A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

(iv) Durability or Permanency – The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

These properties are known as the ***ACID properties***.

Q.3. Define transaction. Discuss ACID properties of a transaction with relevant example. ***(R.G.P.V., May 2018)***

Ans. **Transaction** – Refer to Q.1.

ACID Properties –

Atomicity – Refer to Q.2 (i).

Example – The values of accounts P and Q are ₹1000 and ₹2000, before the execution of transaction T_i. A failure occurs during the execution of transaction T_i, which prevents T_i from completing its execution successfully. Examples of this type of failures are power failures, hardware failures and software errors. Again assume, after the write (P) operation but before the write (Q) operation, the failure occurred. In this condition, the values of accounts P and Q reflected in the database are ₹950 and ₹2000. A result, the system wasted ₹50 of this failure. The sum of P and Q is no longer safe.

Hence, due to failure, the state of the system no longer reflects a real state of the world which the database is considered to capture. This state is known as inconsistent state. It must be confirmed that this type of inconsistencies are invisible in a database system. Although, it should be noted that the system must at some point be in an inconsistent state. Even when transaction T_i is executed to completion, there exists a point at which the value of account P is ₹950 and the value of account Q is ₹2000, which is obviously an inconsistent state. Although, this state is eventually replaced by the consistent state where the value of account P, and the value of account Q are ₹950 and ₹2050. Hence, when the transaction never started or was guaranteed to complete, this type of inconsistent state could be invisible except during the transaction execution. That is the cause for the atomicity necessary. When the atomicity property is present, all actions of the transaction are reflected in the database, or none are.

Durability – Refer to Q.2 (iv).

Example – Once the transaction execution completes successfully. The user who initiated the transaction has been notified that the funds transfer has taken place. It should be the condition that no system failure will result in a loss of data corresponding to this transfer of funds.

Consistency – Refer to Q.2 (ii).

Example – The user may have the consistency criterion that, fund transfers from one bank account to another bank account, should not change the total amount of money in the accounts. For transferring money from one account to another, a transaction must debit one account, temporarily leaving the database inconsistent in a global sense, even though the new account balance can satisfy any integrity constraints with respect to the range of acceptable account balances. If the second account is credited with the transferred amount, then the user's notion of a consistent database is preserved. When a faulty transfer program always credits the second account with ₹1 less than the amount debited from the first account, the database management system may not be expected to detect inconsistencies because of this type of error in the user program's logic.

Isolation – Refer to Q.2 (iii).

Example – When two transactions T_1 and T_2 are executed concurrently then the net effect is guaranteed to be equivalent to executing T_2 followed by executing T_1 or executing T_1 followed by executing T_2. When every transaction

maps a consistent database instance to another consistent database instance, executing many transactions one after the other, results in a consistent last database instance.

Q.4. Explain different states of a transaction. ***(R.G.P.V., June 2014)***

Ans. A transaction must be in one of the following states –

(i) Active – It is the initial state of the transaction. The transaction stays in this state while it is executing.

(ii) Partially Committed – The transaction enters in this state after the final statement has been executed.

(iii) Failed – The transaction enters in this state if the normal execution can no longer proceed.

(iv) Aborted – The transaction enters in this state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

(v) Committed – The transaction enters in this state after successful completion.

Q.5. What are the various states of a transaction ? Explain with a state diagram ?

Ans. **States** – Refer to Q.4.

State Diagram – Fig. 5.2 shows a state transition diagram that describes how a transaction moves through its execution states. A transaction initially goes into an active state and starts execution, where it can issue read and write operation. When the transaction ends, it moves to the partially committed state. At this point, some recovery protocols need to ensure that a system failure will not result in an ability to record the changes of the transaction

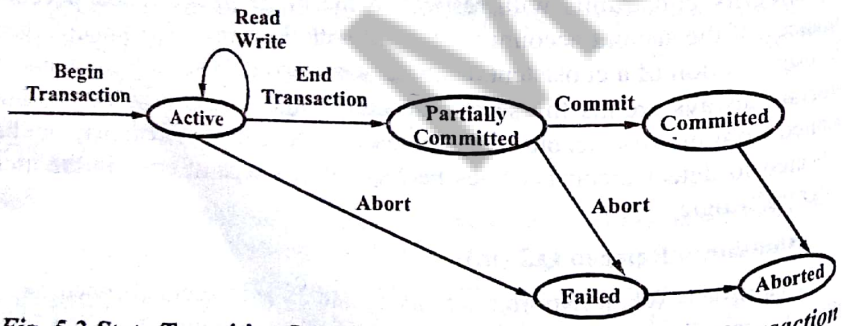


Fig. 5.2 State Transition Diagram Illustrating the States for Transaction Execution

permanently. Once the check is successful, the transaction is said to have reached its commit point and enters the commit state. Once a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database.

However, a transaction can go to the failed state if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its write operations on the database. The terminated state responds to the transaction leaving the system. Failed or aborted transactions may be restarted later either automatically or after being resubmitted by the user as brand new transactions.

Q.6. Define a transaction with a neat sketch discuss the states a transaction can be in then discuss the following with suitable examples –

(i) A read only transaction

(ii) A read write transaction

(iii) An aborted transaction.

(R.G.P.V., Dec. 2014)

Ans. Refer to Q.1, Q.4 and Q.5.

Q.7. Write short note on concurrent operations.

(R.G.P.V., June 2016, Dec. 2016)

Ans. Concurrent operations within a DBMS permit multiple run-units to operate against a single database at the same time in a manner that preserves consistency and integrity. In other words, we can say that concurrent operations let a single run time copy of the DBMS support multiple concurrent update and retrieval transactions from multiple interrogation language run units to the same or different databases without any compromise to database integrity, and with automatic detection and resolution of execution deadlock.

Q.8. What do you mean by concurrent execution ? Discuss its advantages.

Ans. When the transaction processing system give us the permission to run multiple transaction simultaneously, the simultaneously execution of transaction is known as concurrent execution or concurrency. When update the data concurrently to allowing of multiple transaction so its create the some complication and its need extra work for concurrent execution of transaction to ensure consistency of the data.

However, there are two reasons for allowing concurrency –

(i) Improved throughput and resource utilization

(ii) Reduced waiting time.

(i) Improved Throughput and Resource Utilization – A transaction consists of some steps its involve two activity, first is CPU activity and another is I/O activity. In the computer system CPU and disk can be run in parallely.

So I/O activity can be done in parallel with processing at the CPU. So, it can be utilized the parallelism of the CPU and the I/O system to run multiple transaction in parallely. When one disk is in progress of a read or write operation on the one transaction, another transaction can be running in the CPU, where as another disk can be executing a read/write on behalf of a third transaction. Hence increasing the throughput of system and increasing the processor and disk utilization as well as.

(ii) Reduced Waiting Time – A transaction may be short and long running on a system. When transaction is run serially, a short transaction may have to wait for a previous long transaction to till the complete, so in the running transaction it can occurred to unpredictable delays. If the transactions are operating on different parts of the database, so its better run them concurrently sharing the CPU cycles and disk access. Hence, concurrent execution decreases the unpredictable delay and average response time in the running transaction.

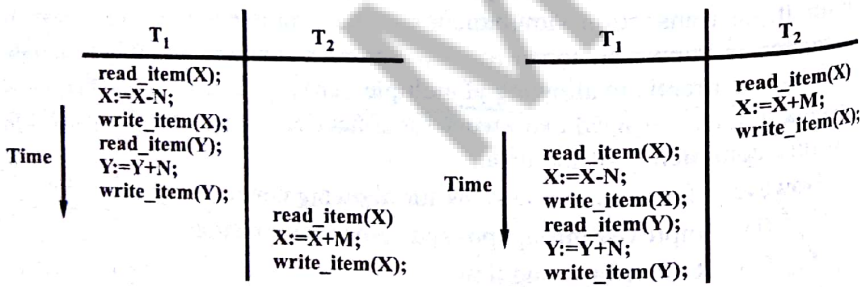
Q.9. What is a schedule ?

Ans. A schedule (or history) S of n transactions T_1, T_2,....., T_n is an ordering of the operations of the transactions subject to the constraint that for each transaction T_i that participates in S, the operations of T_i in S must appear in the same order in which they occur in T_i. However, that operations from other transactions T_j can be interleaved with the operations of T_i in S.

Q.10. Discuss the different types of schedules.

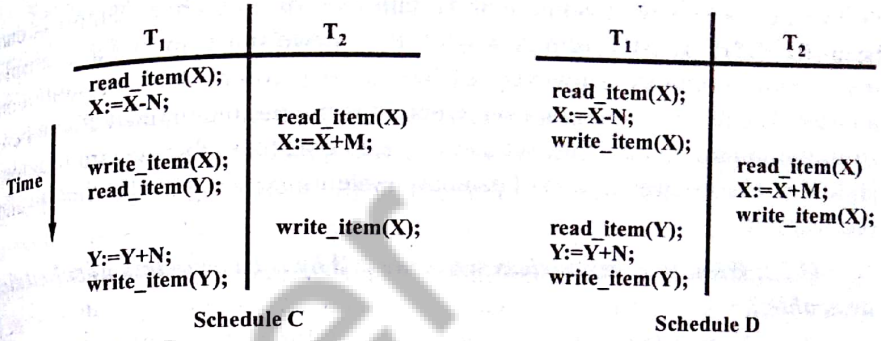
Ans. A schedule S of n transactions T_1, T_2,...., T_n is said to be a ***complete*** schedule if the following conditions hold –

(i) The operations in S are exactly those operations in T_1, T_2,...,T_n, including a commit or abort operation as the last operation for each transaction in the schedule.



(a) Serial Schedule A – T_1 followed by T_2

(b) Serial Schedule B – T_2 Followed by T_1



(c) Two Nonserial Schedules C and D with Interleaving of Operations

Fig. 5.3 Examples of Serial and Nonserial Schedules Involving Transactions T_1 and T_2

(ii) For any pair of operations from the same transaction T_i, their order of appearance in S is the same as their order of appearance in T_i.

(iii) For any two conflicting operations, one of the two must occur before the other in the schedule.

A schedule is called ***serial*** if the operations of each transaction are executed consecutively, without any interleaved operations from other transaction. A schedule is called ***nonserial*** if each sequence interleaves operations from two transactions.

Fig. 5.3 shows four schedules. Schedules A and B are serial because entire transactions are performed in serial order – T_1 and then T_2 in fig. 5.3 (a), and T_2 and then T_1 in fig. 5.3 (b). Schedules C and D in fig. 5.3 (c) are nonserial.

Q.11. What is a recoverable schedule ? Why is recoverability of schedules desirable ?

Ans. A ***recoverable schedule*** is one where, for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j.

If a transaction T_i fails due to any reason, we need to undo the effect of this transaction to ensure the atomicity property of the transaction. In a system that allows concurrent execution, it is also necessary to ensure that any transaction T_j that is dependent on T_i (i.e., T_j has read data written by T_i) is also aborted. Let us, schedule 6 in fig. 5.4, in which transaction T_9 performs only one transaction read (A). Suppose that the system allows T_9 to commit immediately after executing the read (A) instruction. Thus, T_9 commits

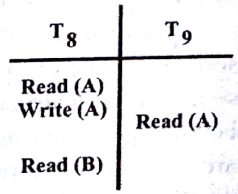


Fig. 5.4 Schedule 6

MSinghal

before T_8 does. Now, assume that T_8 fails before it commits. Since T_9 has read the value of data item A written by T_8, we must abort T_9 to ensure transaction atomicity. However, T_9 has already committed and cannot be aborted. So, there is situation from where it is impossible to recover correctly from the failure of T_8. Thus, schedule 6 is a nonrecoverable schedule, which should not be allowed. Some of database systems require that all schedules be recoverable.

Q.12. What is a cascadeless schedule ? Why is cascadeless of schedules desirable ?

Ans. A cascadeless schedule is one where, for each pair of transaction T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.

Even if a schedule is recoverable, to recover correctly from the failure of a transaction T_i, several transactions are to be rolled back. Some situations occur if transactions have read data written by T_i. For example, assume the partial schedule 7 of fig. 5.5. Transaction T_{21} writes a value of A that is read by transaction T_{22}. Transaction 22 writes a value of A that is read by transaction T_{23}. Assume that, at this point, T_{21} fails. T_{21} must be rolled back. Since T_{22} is dependent on T_{21}, T_{22} must be rolled back. Since T_{23} is dependent on T_{22}, T_{23} must be rolled back. This phenomenon, in which a single transaction failure leads to a series of transaction rollback, is called cascading rollback.

T_{21}	T_{22}	T_{23}
Read (A)		
Read (B)		
Write (A)		
	Read(A)	
	Write(A)	
		Read A

Fig. 5.5 Schedule 7

Cascading rollback is undesirable, since it leads to the undoing of a significant amount of work. It is desirable to restrict the schedules to those where cascading rollbacks cannot occur.

Q.13. Write short note on serializability. ***(R.G.P.V., June 2006)***

Or

Explain serializability. ***(R.G.P.V., Dec. 2015)***

Ans. The concept of serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedules.

A schedule S of n transactions is ***serializable*** if it is equivalent to some serial schedule of the same n transactions. Each serial schedule consists of a sequence of instructions from various transactions, where the instructions belonging to one single transaction appear together in that schedule. So, there are n ! possible serial schedules of n transactions and much more possible nonserial schedules.

Saying that a nonserial schedule S is serializable is equivalent to saying that it is correct, because it is equivalent to a serial schedule, which is considered correct.

Q.14. Discuss the concept of view serializability.

Ans. Consider two schedules S and S' with the same set of transactions participating in both schedules. The schedules S and S' are said to be view equivalent if following three conditions are met –

(i) For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i in schedule S' must read the initial value of Q.

(ii) For each data item Q, if transaction T_i executes read (Q) in schedule S, and if that value was produced by a write (Q) operation executed by transaction T_j, then the read (Q) operation of transaction T_i in schedule S' must also read the value of Q that was produced by the same write (Q) operation of transaction T_j.

(iii) For each data item Q, the transaction (if any) that performs the final write (Q). Operation in schedule S must perform the final write (Q) operation in schedule S'.

Conditions (i) and (ii) ensure that each transaction reads the same values in both schedules and, therefore, performs the same computation. Condition (iii), coupled with conditions (i) and (ii), ensures that both schedules result in the same final system state.

The concept of view equivalence loads to the concept of view serializability. A schedule S is view serializable if it is view equivalent to a serial schedule.

For example, fig. 5.6 shows schedule 5, which is view serializable. It is view equivalent to the serial schedule $<T_3, T_4, T_6>$, since the one read (Q) instruction reads the initial value of Q, and T_6 performs the final write of Q.

T_3	T_4	T_6
Read (Q)		
	Write (Q)	
Write (Q)		Write (Q)

Fig. 5.6 Schedule 5 – A View Serializable Schedule

Every conflict-serializable schedule is view serializable, but its vice-versa is not true.

Q.15. Discuss the concept of conflict serializability.

Or

Explain the concept of conflict serializability. Define conflicting instructions and non-conflicting instructions. ***(R.G.P.V., Dec. 2017)***

Ans. Consider a schedule S in which there are two consecutive instructions I_i and I_j, of transactions T_i and T_j respectively ($i \neq j$). If I_i and I_j refer to

different data items, then I_i and I_j can be swapped without affecting the results of any instruction in the schedule. However, if I_i and I_j refer to the same data item Q, then the order of steps may matter. There are four cases to be considered while dealing with only read and write instructions –

(i) I_i = read (Q), I_j read(Q). The order of I_i and I_j does not matter, since the same value of Q is read by T_i and T_j, regardless of the order.

(ii) I_i = read (Q), I_j = write (Q). If I_i comes before I_j, then T_i does not read the value of Q that is written by T_j in instruction. If I_j comes before I_i, then T_i reads the value of Q that is written by T_j. Thus, the order of I_i and I_j matters.

(iii) I_i = write (Q), I_j = read (Q). The order of I_i and I_j matters as in case (ii).

(iv) I_i = write (Q), I_j = write (Q). The order of these instructions does not affect either T_i or T_j, since both are write operations. However, the value obtained by the next read (Q) instruction of S is affected, since the result of only the latter of the two write instructions is preserved in the database. If there is no other write (Q) instruction after I_i and I_j in S, then the order of I_i and I_j directly affects the final value of Q in the database state that results from schedule S.

We say that I_i and I_j conflict if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.

For example, consider the schedule 1 in fig. 5.7. The write (A) instruction of T_1 conflicts with the read (A) instruction of T_2. However, the write (A) instruction of T_2 does not conflict with the read (B) instruction of T_1, because the two instructions access different data items.

Let I_i and I_j be consecutive instructions of a schedule S. If I_i and I_j are instructions of different transactions and I_i and I_j do not conflict, the order of I_i and I_j can be swapped to produce a new schedule S'. It is expected that S is equivalent to S', since all instructions appear in the same order in both schedules except for I_i and I_j, whose order does not matter.

T_1	T_2
Read (A)	
Write (A)	
	Read (A)
	Write (A)
Read (B)	
Write (B)	
	Read (B)
	Write (B)

Fig. 5.7 Schedule 1 – Showing Only the Read and Write Instructions

T_1	T_2
Read (A)	
Write (A)	
	Read (A)
Read (B)	
Write (B)	
	Write (A)
	Read (B)
	Write (B)

Fig. 5.8 Schedule 2 – Schedule 1 after Swapping of a Pair of Instruction

Since the write (A) instruction of T_2 in schedule 1 does not conflict with the read (B) instruction of T_1, we can swap these instructions to generate an equivalent schedule, schedule 2 in fig. 5.8. Schedules 1 and 2 both produce the same final system state.

Now, continue to swap nonconflicting instructions –

(i) Swap the read (B) instruction of T_1 with the read (A) instruction of T_2.

(ii) Swap the write (B) instruction of T_1 with the write (A) instruction of T_2.

(iii) Swap the write (B) instruction of T_1 with the read (A) instruction of T_2.

The final result of these swaps is schedule 3 as shown in fig. 5.9. It is a serial schedule. Thus, schedule 1 is equivalent to a serial schedule. This equivalence implies that, regardless of the initial system state, schedule 1 will produce the same final state as will some serial schedule.

T_1	T_2
Read (A)	
Write (A)	
Read (B)	
	Read (A)
	Write (A)
	Read (B)
	Write (B)

Fig. 5.9 Schedule 3 – A Serial Schedule Equivalent to Schedule 1

T_3	T_4
Read (Q)	
	Write (Q)
Write (Q)	

Fig. 5.10 Schedule 4

If a schedule S can be transformed into a schedule S' by a series of swaps of nonconflicting instructions, the S and S' are ***conflict equivalent.*** The concept of conflict equivalence leads to the concept of conflict serializability. A schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

Fig. 5.10 shows schedule 4, which consists of read and write operations of transactions T_3 and T_4. This schedule is not conflict serializable, since it is not equivalent to either the serial schedule $<T_3, T_4>$ or the serial schedule $<T_4, T_3>$. It is possible that two schedules produce the same result, but are not conflict equivalent.

Q.16. What is serializable schedule ? Describe with suitable example the types of serializable schedules. ***(R.G.P.V., June 2014)***

Ans. Refer to Q.13, Q.14 and Q.15.

Not Wanter

Q.17. What is serializability ? Write an algorithm for testing conflict serializability of a schedule. ***(R.G.P.V., Dec. 2006)***

Ans. **Serializability** – Refer to Q.13.

Algorithm for Testing Conflict Serializability of a Schedule – When designing concurrency control schemes, we must show that schedules generated by the scheme are serializable. Algorithm 5.1 can be used to test a schedule for conflict serializability. The algorithm looks at only the read_item and write_item operations in a schedule to construct a precedence graph (or serialization graph), which is a directed graph G = (V, E), where V is a set of nodes $\{T_1, T_2, ...,T_n\}$ and E is a set of directed edges $\{e_1, e_2,...,e_m\}$. There is one node in the graph for each transaction T_i in the schedule. Each edge e_i in the graph is of the form $(T_j \rightarrow T_k)$ $1 \le j \le n$, $1 \le k \le n$, where T_j is the starting node of e_i and T_k is the ending node of e_i. Such an edge is created if one of the operations in T_j appears in the schedule before some conflicting operation in T_k.

Algorithm 5. 1 – Testing conflict serializability of a schedule S.

(i) For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.

(ii) For each case in S where T_j executes a read_item (X) after T_i executes a write_item (X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.

(iii) For each case in S where T_j executes a write_item(X) after T_i executes a read_item (X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.

(iv) For each case in S where T_j executes a write_item (X) after T_i executes a write_item (X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.

(v) The schedule S is serializable if and only if the precedence graph has no cycle.

The precedence graph is constructed as described in algorithm 5.1. If there is a cycle in the precedence graph, schedule S is not (conflict) serializable. If there is no cycle, S is serializable. A cycle in a directed graph is a sequence of edges $C = ((T_j \rightarrow T_k), (T_k \rightarrow T_p),.....,(T_i \rightarrow T_j))$ with the property that the starting node of each edge except the first edge is the same as the ending node of the last edge i.e., the sequence starts and ends at the same node.

In the precedence graph, an edge from T_i to T_j means that transaction T_i must come before transaction T_j in any serial schedule that is equivalent to S, because two conflicting operations appear in the schedule in that order. If there is no cycle in the precedence graph, we can create an equivalent serial schedule's that is equivalent to S, by ordering the transactions that participate in S as follows – Whenever an edge exists in the precedence graph from T_i to T_j, T_i must appear before T_j in the equivalent serial schedule S'. The edges $T_i \rightarrow T_j$ can optionally be labeled by the name(s) of the data item (s) that led to creating the edge.

In general, several serial schedules can be equivalent to S if the precedence graph for S has no cycle. However, if the precedence graph has a cycle, it is easy to show that we cannot create any equivalent serial schedule, so S is not serializable. The precedence graph created for schedules A to D, respectively, of fig. 5.3 appear in fig. 5.11 (a) to (d). The graph for schedule C has a cycle, so it is not serializable. The graph for schedule D has no cycle, so it is serializable, and the equivalent serial schedule is T_1 followed by T_2. The graphs for schedules A and B have no cycles because the schedules are serial and hence serializable.

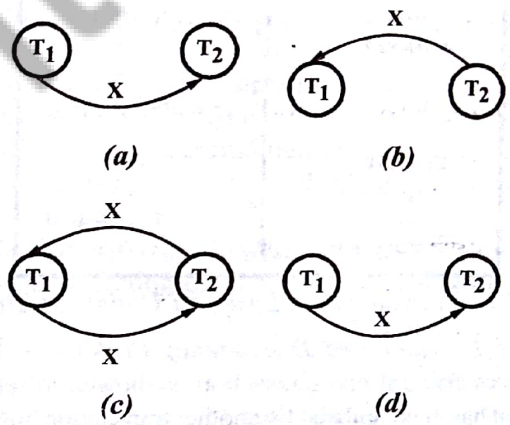


Fig. 5.11 Construction of the Precedence Graph for Schedules A to D of Fig. 5.3

Testing for view serializability is rather complicated. In fact, it has been shown that the problem of testing for view serializability is itself NP-complete. Thus, there exists no efficient algorithm to test for view serializability.

Q.18. Describe about the testing of serializability. (R.G.P.V., Dec. 2012)

Ans. Refer to Q.17.

Q.19. What is data tuning ? ***(R.G.P.V., Dec. 2014)***

Ans. Database performance tuning refers to a set of activities and procedures designed to reduce the response time of the database system, that is, to ensure that an end-user query is processed by the DBMS in the minimum amount of time. The time required by a query to return a result set depends on many factors, that is called "data tuning".

Asinvente

Q.20. What are the problems of concurrent transactions of database ? Explain.

Ans. Transactions submitted by the various users may execute concurrently and may access and update the same database items. If this concurrent execution is uncontrolled, it may lead to problems. The three problems are as follows –

(i) The Lost Update Problem – Consider the situation shown in fig. 5.12. This figure is meant to be read as follows – Transaction A retrieves some tuple t at time t_1; transaction B retrieves that same tuple t at time t_2; transaction A updates the tuple at time t_3; and transaction B updates the same tuple at time t_4. Tran-saction A's update is lost at time t_4, because transaction B overwrites it without even looking at it.

Transaction A	Time	Transaction B
—		—
—	t_1	—
Retrieve t		—
—	t_2	—
—		Retrieve t
—	t_3	—
Update t		—
—	t_4	—
—		Update t
—		—

Fig. 5.12 Transaction A Loses an Update at Time T_4

(ii) The Uncommitted Dependency Problem – The uncommitted dependency problem arises if one transaction is allowed to retrieve – or, worse, update – a tuple that has been updated by another transaction but not yet committed by that other transaction. For if it has not yet been committed, there is always a possibility that it never will be committed but will be rolled back instead – in which case the first transaction will have seen some data that now no longer exists.

Consider figs. 5.13 and 5.14.

Transaction A	Time	Transaction B
—		—
—	t_1	—
—		Update t
—	t_2	—
Retrieve t		—
—	t_3	—
—		Rollback
—		

Fig. 5.13 Transaction A becomes Dependent on an Uncommitted Change at Time t_2

Transaction A	Time	Transaction B
—		—
—	t_1	—
—		Update t
—	t_2	—
Update t		—
—	t_3	—
—		Rollback
—		

Fig. 5.14 Transaction A Updates an Uncommitted Change at Time t_2, and Loses that Update at Time t_3

In the first example (fig. 5.13), transaction A sees an uncommitted update at time t_2. That update is then undone at time t_3. Transaction A is therefore operating on a false assumption – namely, the assumption that tuple t has the value seen at time t_2, whereas infact it has whatever it had prior to time t_1. As a result transaction A may produce incorrect output.

The second example (fig. 5.13) is even worse. Not only does transaction A become dependent on an uncommitted change at time t_2, but it actually loses an update at time t_3 because the rollback at time t_3 causes tuple t to be restored to its value prior to time t_1.

(iii) The Inconsistent Analysis Problem – Fig. 5.15 shows two transactions A and B operating on account (ACC) tuples – Transaction A is summing account balances, transaction B is transferring an amount 10 from account 3 to account 1. The result produced by A, 110, is obviously incorrect; if A were to go on to write that result back into the database, it will actually leave the database in an inconsistent problem.

Acc 1	Acc 2	Acc 3
40	50	30

Transaction A	Time	Transaction B
—		—
—	t_1	—
Retrieve Acc 1 : sum = 40		—
Retrieve ACC 2 : sum = 90	t_2	—
—	t_3	—
—		—
—	t_4	Retrieve Acc 3
—	t_5	—
—		Update Acc 3 : 30 → 20
—	t_6	—
—		Retrieve Acc 1
—	t_7	—
—		Update Acc 1 : 40 → 50
—	t_8	—
Retrieve Acc 3 : sum = 110, not 120		Commit

Fig. 5.15 Transaction A Performs an Inconsistent Analysis

CONCURRENCY CONTROL – LOCK MANAGEMENT, SPECIALIZED LOCKING TECHNIQUES, CONCURRENCY CONTROL WITHOUT LOCKING, PROTECTION AND SECURITY

Q.21. Explain the concept of lock for concurrency control.

Or

What is concurrency control ? How is it implemented in DBMS? Explain. (R.G.P.V., June 2010, Dec. 2012)

Ans. The coordination of the simultaneous execution of transaction in a multiuser database system is known as ***concurrency control.***

The objective of concurrency control is to ensure, the serializability of transactions in a multiuser database environment. One way to ensure serializability is to require that data items be accessed in a mutually exclusive manner. That is, while one transaction is accessing a data item, another transaction can modify that data item. The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item. A ***lock*** is a variable associated with a data item that describes the status of the data item with respect to possible operations that can be applied to it manipulating the value of a lock is called ***locking.*** Generally, there is a one lock for each data item in the database. Locks are used as a means of synchronizing the access by concurrent transactions to the database items.

There are two types of lock –

(i) Exclusive Lock – The exclusive lock is also called an update or a write lock. The intention of this mode of locking is to provide exclusive use of the data item to one transaction. If a transaction T_i has obtained an exclusive-mode lock (denoted by X) on item Q, then T_i can both read and write Q.

(ii) Shared Lock – The shared lock is also called a read lock. The intention of this mode of locking is to ensure that the data – item does not undergo any modifications. Any number of transactions can concurrently lock and access a data-item in the shared mode, but none of these transactions can modify the data-item. If a transaction T_i has obtained a shared-mode lock (denoted by S) on item Q, then T_i can read, but cannot write Q.

We need that every transaction request a lock in an appropriate mode on data item Q, depending on the types of operations that it will perform on Q. The transaction makes the request to the concurrency-control manager. The transaction can proceed with the operation only after the concurrency-control manager grants the lock to the transaction.

A compatibility function on a set of lock modes can be defined as follows – Let A and B represent arbitrary lock modes. Suppose that a transaction T_i requests a lock of mode A on item Q on which transaction T_j ($T_i \neq T_j$) currently holds a lock of mode B. If transaction T_i can be granted a lock on Q immediately, inspite of the presence of the mode B lock, then mode A is compatible with mode B. This function can be represented by a matrix. The compatibility relation between the two modes of locking is shown in the matrix COMP of fig. 5.16. An element comp (A, B) of the matrix has the value true if and only if mode A is compatible with mode B.

	S	X
S	True	False
X	False	False

Fig. 5.16 Lock-compatibility Matrix Comp

A shared mode is compatible with shared mode, but not with exclusive mode. At any time, several shared-mode locks can be held simultaneously by different transactions on a particular data item. A subsequent exclusive mode lock request has to wait until the currently held shared-mode locks are released.

A transaction requests a shared lock on data item Q by executing the lock-S (Q) instruction. Similarly, a transaction requests an exclusive lock through the lock–X(Q) instruction. A transaction can unlock a data item Q by the unlock (Q) instruction.

When access a data item, transaction T_i must be first lock that item. Then the data item is already locked by another transaction in an incompatible mode, the concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. So, T_i is made to wait until all incompatible locks held by other transactions have been released.

Q.22. Describe the three most common concurrent transaction execution problems. Explain how concurrency control can be used to avoid those problems. (R.G.P.V., Dec. 2011, 2013)

Ans. Refer to Q.20 and Q.21.

Q.23. What DBMS component is responsible for concurrency control ? How is this feature used to resolve conflicts ? (R.G.P.V., Dec. 2011)

Ans. The component of DBMS which is responsible for establishing the order in which the concurrent transaction operations are executed is scheduler. It is a special DBMS program. To ensure serializability and isolation of transactions, the scheduler interleaves the execution of database operations. The scheduler bases its actions on concurrency control algorithms such as locking or time stamping methods to determine the proper order.

In addition, data isolation is facilitated by the scheduler to ensure that two transactions do not update the same data element at the same time. Database

operations may need READ and/or WRITE actions that produce conflicts. As an example, table 5.1 depicts the possible conflict scenarios when two transactions, T_1 and T_2 are executed concurrently over the same data. It is clearly shown by table 5.1 that two operations are in conflict when they access the same data and at least one of them is a WRITE operation.

Table 5.1 Read/Write Conflict Scenarios

T_1	T_2	Result
Read	Read	No conflict
Read	Write	Conflict
Write	Read	Conflict
Write	Write	Conflict

Many methods have been suggested to schedule the execution of conflicting operations in concurrent transactions. Those methods have been categorized at locking, time stamping, and optimistic. Locking methods are used most frequently.

Q.24. What are the three kinds of intent locks ? (R.G.P.V., Dec. 2014)

Ans. According to intent locking protocol no transaction is allowed to acquire a lock on a tuple before first acquiring a lock on the relvar that contains it. The three kinds of intent locks are intent shared (IS), intent exclusive (IE) and shared intent exclusive (SIX) locks.

Q.25. Discuss on two-phase locking protocol. (R.G.P.V., Dec. 2012)

Or

Explain two phase locking protocol with example. (R.G.P.V., June 2014)

Or

Explain the lock based protocols for concurrency control. (R.G.P.V., June 2011)

Or

Discuss two phase locking protocol with suitable example. (R.G.P.V., Dec. 2014)

Or

Write short note on two phase locking protocol. (R.G.P.V., May 2018)

Ans. One protocol that ensures serializability is the two-phase locking protocol. A transaction is said to follow the two-phase locking protocol if all locking operations (read_lock write_lock) precede the first unlock operation in the transaction. This transaction can be divided into two phase –

(i) Expanding/Growing (first) phase – During this phase, a transaction can obtain locks but cannot release any lock.

(ii) Shrinking (second) Phase – During this phase, a transaction can release locks, but cannot obtain any new locks.

If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, and downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase. So, a read-lock (X) operation that downgrades an already held write lock on X can appear only in the shrinking phase.

Transactions T_1 and T_2 in fig. 5.17 follow two-phase locking protocol.

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y);	unlock(X);
read_item(X);	read_item(Y);
X := X + Y;	Y := X + Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Fig. 5.17 Schedule Under Two-phase Locking

It can be proved that, if every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable, obviating the need to test for serializability of schedules any more. The locking mode, by enforcing two-phase locking rules, also invoked serializability.

Two-phase locking may limit the amount of concurrency that can occur in a schedule. This is because a transaction T may not be able to release an item X after it is through using it if T must lock an additional item Y later on; or conversely. So, X must remain locked by T until all items that the transaction needs to read or write have been locked; only then can X be released by T. Meanwhile another transaction seeking to access X may be forced to wait, even though T is done with X; conversely, if Y is locked earlier than it is required, another transaction seeking to access Y is forced to wait even though T is not using Y yet. This is the price for guaranteeing serializability of all schedules without having to check the schedules themselves.

Q.26. What is strict two-phase locking ? What are the advantages and disadvantages of strict two-phase locking ?

Ans. A variation of two-phase locking is known as the strict two-phase locking protocol, which guarantees strict schedules. This protocol requires not only that locking be two phase, but also that all exclusive-mode locks taken by a transaction be held until that transaction commits. This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.

MSinventor

Advantages –

(i) The strict two-phase locking protocol ensures that the schedules involving transactions using this protocol will always be serializable.

(ii) The strict two-phase locking protocol avoids cascading rollbacks.

Disadvantages –

(i) It creates the problem of deadlock.

(ii) It also creates the problem of starvation.

Q.27. What are the various concurrency control techniques ? Explain. ***(R.G.P.V., June 2006)***

Or

List the commonly used concurrency control techniques. ***(R.G.P.V., Dec. 2014)***

Or

List the two commonly used concurrency control techniques. ***(R.G.P.V., June 2015)***

Ans. Refer to Q.21, Q.25, and Q.26.

Q.28. Write short note on locking techniques. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.25 and Q.26.

Q.29. Explain how strict 2-phase locking is implemented. Show them with an example. ***(R.G.P.V., June 2017)***

Ans. **Strict 2-phase Locking Protocol** – Refer to Q.26.

Implementation Example – See fig. 5.18.

```
T1
read_lock(X);
write_lock(X);
read_lock (Y);
write_lock(Y);
read_item(X);
X : = X – 500;
read_item(Y);
Y : = Y + 500;
write_item(X);
write_item(Y);
unlock_write(Y);
unlock_read(Y);
unlock_write(X);
unlock_read(X);
```

Fig. 5.18 Strict Two-phase Locking Transaction

Fig. 5.18 shows strict two-phase locking protocol in a transaction in which all exclusive-mode locks taken by transaction (i.e. read_lock and write_lock).

Q.30. What are two pitfalls (problems) of lock-based protocols ? ***(R.G.P.V., Dec. 2014)***

Ans. Following are the two problems of lock-based protocols –

(i) The schedule may create deadlocks.

(ii) The resulting transaction schedule may not be serialize.

Q.31. Explain timestamp based protocol for concurrency control. ***(R.G.P.V., June 2011)***

Ans. The idea for this scheme is to order the transactions based on their timestamps. A schedule in which the transactions participate is then serializable, and the equivalent serial schedule has the transactions in order of their timestamp values. This is called ***timestamp ordering (TO).*** In timestamp ordering, the schedule is equivalent to the particular serial order corresponding to the order of the transaction timestamps. The algorithm must ensure that, for each item accessed by conflicting operations in the schedule, the order in which the item is accessed does not violate the serializability order. To do this, the algorithm associates with each database item X two timestamp (TS) values –

(i) Read_TS(X) – It stands for ***read timestamp*** of item X. This is the largest timestamp among all the timestamps of transactions that have successfully read item X – that is, read_TS(X) = TS(T), where T is the youngest transaction that has read X successfully.

(ii) Write_TS(X) – It stands for ***write timestamp*** of item X. This is the largest of all the timestamps of transactions that have successfully written item X – that is, write_TS(X) = TS(T), where T is the youngest transaction that has written X successfully.

(i) Basic Timestamp Ordering – Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with the read_TS(X) and write_TS(X) to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp. If T is aborted and rolled back, any transaction T_1 that may have used a value written by T must also be rolled back. Similarly, any transaction T_2 that may have used a value written by T_1 must also be rolled back, and so on. This effect is known as ***cascading rollback*** and is one of the problems associated with basic TO, since the schedules produced are not recoverable. An additional protocol must be enforced to ensure that the schedules are recoverable, cascadeless or strict. The basic TO algorithm is described below. The concurrency control algorithm must check whether conflicting operations violate the timestamp ordering in

Visin Vihar

the following two cases –

(a) Transaction T issues a write_item(X) operation –

(1) If read_TS(X) > TS(T) or if write_TS(X) >TS(T), then abort and roll back T and reject the operation. This should be done because some younger, transaction with a timestamp greater than TS(T), and hence after T in the timestamp ordening, has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering.

(2) If the condition in part (1) does not occur, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

(b) Transaction T issues a read_item(X) operation –

(1) If write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with timestamp greater than TS(T), and hence after T in the timestamp ordering, has already written the value of item X before T had a chance to read X.

(2) If write_TS(X) < = TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

So, whenever the basic TO algorithm detects two conflicting operations that occur in the incorrect order, it rejects the later of the two operations by aborting the transaction that issued it. The schedules produced by basic TO are so guaranteed to be conflict serializable.

(ii) Strict Timestamp Ordering – A variation of basic TO called ***strict TO*** ensures that the schedules are both strict (for easy recoverability) and (conflict) serializable. In this variation, a transaction T that issues a read_item (X) or write_item(X) such that TS(T) > write_TS(X) has its read or write operation delayed until the transaction T' that wrote the value of X (hence TS(T') = write_TS(X)) has committed or aborted. To implement this algorithm, it is necessary to simulate the locking of an item X that has been written by transaction T until T' is either committed or aborted. This algorithm does not cause deadlock, since T waits for T' only if TS(T) > TS(T').

Q.32. What is serializability ? How can serializability be ensured ? Do you need to restrict ? ***(R.G.P.V., June 2017)***

Ans. **Serializability** – Refer to Q.13.

There are following protocols which ensures serialisability.

(i) Two Phase Locking Protocol – Refer to Q.25.

(ii) Timestamp Based Protocol – Refer to Q.31.

Serializability ensures that the transactions are being executed successfully. So, to ensure successful execution of transaction we need to restrict serializability.

Q.33. Discuss on two-phase locking protocol and timestamp based protocol. ***(R.G.P.V., June 2010)***

Ans. **Two-phase Locking Protocol** – Refer to Q.25.

Timestamp Based Protocol – Refer to Q.31.

Q.34. Concurrent execution of transaction to ensure serializability ? Justify your answer. Give an example of transactions and how you can force serializability in those transactions. ***(R.G.P.V., June 2017)***

Ans. Usually transaction processing systems allows concurrent multiple transitions. But if multiple transactions update same data concurrently then this will loads to inconsistency of data. So, only the concurrent execution of transaction will not ensure serializability. To ensure serializability, the DBMS must control the concurrent execution of transactions. There are certain control mechanisms like two-phase locking protocols or timestamp based protocols these ensures serializability in concurrent transactions.

Let us take an example –

Let there is ₹10000 in account A and A is joint account of two persons. Now, person1 wants to withdraw ₹4000 from account A and at the same time person 2 wants to withdraw ₹5000 from account A. Their concurrent transactions T_1 and T_2 are shown in table 5.2.

Table 5.2

T_1		T_2	
Transition	***Calculation***	***Transition***	***Calculation***
read (A) A = A – 5000 write (A)	A = 10000 A = 5000 A = 5000 (saved)	read (A) A = A – 4000 write (A)	A = 10000 A = 6000 A = 6000 (saved)

According to the table after withdrawing ₹5000 and ₹4000 concurrently from account A the remaining money is ₹6000, this is what the problem of concurrency without serializability. To remove this inconsistency of data, the transactions must be serialize. Locks can force serializability in these transaction. After adding locks the transactions are shown in fig. 5.19.

T_1	T_2
read_ lock(A)	read_ lock(A)
write_ lock(A)	write_ lock(A)
read(A)	read(A)
A = A – 5000	A = A – 4000
write(A)	write(A)
Unlock_ write(A)	Unlock_ write(A)
Unlock_ read(A)	Unlock_ read(A)

Fig. 5.19

So, when there is a read lock in any transaction then no other transaction can read those data. This is also for write lock. So, this is how the locks can force the serializability in the transactions.

Q.35. What do you mean by time stamping protocol for concurrency control ? Discuss multi-version scheme of concurrency control also.
(R.G.P.V., Dec. 2013)

Ans. **Time Stamping Protocol** – Refer to Q.31.

Multi-version Scheme – In multiversion concurrency control the old values of a data item are kept when the item is updated. When a transaction needs access to an item, an appropriate version is selected to maintain the serializability of the currently executing schedule, if possible.

Multiversion concurrency control schemes based on timestamp ordering and 2PL are discussed below –

(i) Multiversion Timestamp Ordering – Here, several versions X_1, X_2, ..., X_K of each data item X are maintained. The value of version X_i and the following two timestamps are kept for each version –

(a) Read_TS(X_i) – It is pronounced as the read timestamp of X_i. It is the largest of all the timestamps of transactions that have successfully read version X_i.

(b) Write_TS (X_i) – It is pronounced as the write timestamp of x_i. It is the timestamp of the transaction that wrote the value of version x_i.

When a transaction T executes a write_item (X) operation, a new version X_{K+1} of item X is created with both the read_TS (X_{K+1}) and the write_TS (X_{K+1}) set to TS(T). Likewise, when a transaction T reads the value of version X_i, the value of read_TS (X_i) is set to the larger of the current read_TS (X_i) and TS (T).

The following two rules are used to ensure serializability –

(i) If transaction T issues a write_item (X) operation and version i of X has the highest write_TS (X_i) of all versions of X that is also less than or equal to TS (T), and read_TS (X_i) > TS (T), then abort and roll back transaction T. Otherwise, create a new version x_i of X with read_TS (X_j) = write_TS (X_j) = TS (T).

(ii) If transaction T issues a read_item (X) operation, find the version i of X having the highest write_TS (X) of all versions of X that is also less than or equal to TS (T). Then return the value of X_i to transaction T, and set the value of read_TS (X_i) to the larger of TS (T) and the current read_TS (X_i).

(ii) Multiversion Two-phase Locking – In this scheme, there are three modes for an item – read, write and certify. So, the state of LOCK(X) for an item X can be one of the following – read-locked, write-locked, certify-locked, or unlocked. In the standard locking scheme with only one read and write locks a write lock is an exclusive lock. The relationship between read and write locks in the standard scheme can be described by means of the lock compatibility as shown in fig. 5.20 (a). An entry of yes indicates that, if a transaction T holds the type of lock specified in the column header on item X and if transaction T' requests the type of lock specified in the column header on item X, then T' can get the lock because the locking modes are compatible. In contrast, an entry of no in the table means that the locks are not compatibe, so T' must wait until T releases the lock.

	Read	Write
Read	yes	no
Write	no	no

(a) Compatibility Table for Read/write Locking Scheme

	Read	Write	Certify
Read	yes	no	no
Write	yes	no	no
Certify	no	no	no

(b) Compatibility Table for Read/write/certify Locking scheme

Fig. 5.20 Lock Compatibility Tables

In the standard locking scheme, once a transaction gets a write lock on an item, no other transactions can access that item. The idea behind multiversion 2 PL is to permit other transactions T' to read an item X while a single transaction T holds a write lock on X. This is accomplished by allowing two versions for each item X. One version must always have been written by some committed transaction. The second version X' is created when a transaction T obtains a write lock on the item. Other transactions can continue to read the committed version of X while T holds the write lock. Transaction T can write the value of X' as required, without affecting the value of the committed version X. However, once, T is ready to commit, it must get a certify lock on all items that it currently holds write locks on before it can commit. The certify lock is not compatible with read locks, so the transaction

may have to delay its commit until all its write-locked items are released by any reading transactions in order to get the certify locks. Once the certify locks are obtained, the committed version X of the data item is set to the value of version X', version X' is discarded, and the certify locks are then released. Fig. 5.20 (b) shows the lock compatibility table for this scheme

Q.36. Explain 2-phase commitment protocol and the behavior of this protocol during lost messages and site failures. ***(R.G.P.V., Dec. 2013)***

Ans. It is necessary to have a two-level recovery mechanism to maintain the atomicity of a multidatabase transaction. A global recovery manager, or coordinator, is required to maintain information required for recovery, in addition to the local recovery managers and the information they maintain (log, tables). Usually, the coordinator follows a protocol known as the two-phase commit protocol, whose two phases are as follows –

Phase 1 – If all participating databases signal the coordinator that the part of the multidatabase transaction involving each has concluded, the coordinator sends a message "prepare for commit" to each participant to get ready for committing the transaction. Each participating database receiving that message will force-write all log records and required information for local recovery to disk and then send a "ready to commit" or "OK" signal to the coordinator. If the force-writing to disk fails or the local transaction cannot commit for some reason, the participating database sends a "cannot commit" or "not OK" signal to the coordinator. If the coordinator does not receive a reply from a database within a certain time out interval, it assumes a "not OK" response.

Phase 2 – When all participating databases reply "OK", and the coordinator's vote is also "OK", the transaction is successful, and the coordinator sends a "commit" signal for the transaction to the participating databases. Recovery from failure is now possible because all the local effects of the transaction and information required for local recovery have been recorded in the logs of the participating databases. Each participating database completes transaction commit by writing a [commit] entry for the transaction in the log and permanently updating the database if required. In contrast, if one or more of the participating databases or the coordinator have a "not OK" response, the transaction has failed, and the coordinator sends a message to "roll back" or UNDO the local effect of the transaction to each participating database. This is accomplished by undoing the transaction operations, using the log.

The total effect of the two-phase commit protocol is that either all participating databases commit the effect of the transaction or none of them do. In case any of the participants fails, it is always possible to recover to a

state where either the transaction is committed or it is rolled back. A failure during or before phase 1 usually needs the transaction to be rolled back, whereas a failure during phase 2 refers to a successful transaction can recover and commit.

Q.37. Discuss the problem of deadlock and starvation, and the different approaches to dealing with these problems.

Or

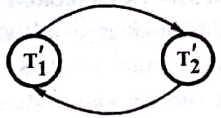
What is deadlock ? Discuss the different ways of preventing deadlock and detecting deadlock. ***(R.G.P.V., Dec. 2017)***

Ans. **Deadlock** – A system is in a deadlock state if each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set. So, each transaction in the set is on a waiting queue, waiting for one of the other transactions in the set to release the lock on an item.

Fig. 5.21 shows that two transactions T'_1 and T'_2 are deadlocked in a partial schedule T'_1 is on the waiting queue for X , which is locked by T'_2 and T'_2 is on the waiting queue for Y which is locked by T'_1.

Time	T'_1	T'_2
↓	Read_lock(Y); Read_item(Y);	
		Read_lock(X); Read_item(X);
	Write_lock(X);	
		Write_lock(Y);

(a) A Partial Schedule of T'_1 and T'_2 in Deadlock State



(b) A Wait-for Graph for the Partial Schedule in (a)

Fig. 5.21 Illustration of Deadlock Problem

Deadlock Prevention Protocols – The way to prevent deadlock is to use a deadlock prevention protocol. One deadlock prevention protocol requires that every transaction locks all the items it needs in advance i.e., before execution. If any of the items cannot be locked, none of the items are locked. Rather, the transaction waits and then tries again to lock all the items it needs. This solution has two main disadvantages –

(i) It is very hard to predict, before the transaction begins, what data items need to be locked.

(ii) Data-item utilization may be very low, since many of the data items may be locked but unused for a long time.

Another approach is to impose an ordering of all data-items, and to require that a transaction lock data item in a sequence consistent with the ordering.

The second approach for preventing deadlocks is to use preemption and transaction rollbacks. In preemption, when a transaction T_2 requests a lock that transaction T_1 holds, the lock granted to T_1 may be preempted by rolling back of T_1, and granting of the lock to T_2. To control the preemption, we assign a unique timestamp to each transaction. The system uses these timestamps to decide whether a transaction should wait or roll back. Locking is still used for concurrency control. If a transaction is rolled back, it retains its old timestamp when restarted. Two different deadlock prevention schemes using timestamps are as follows –

(i) Wait-die – It is a nonpreemptive technique. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than of T_j (i.e., T_i is older then T_j). Otherwise, T_i is rolled back (dies).

(ii) Wound-wait – This is a preemptive technique. It is a counterpart to the wait-die scheme. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j (i.e., T_i is younger than T_j). Otherwise, T_j is rolled back (T_j is wounded by T_i).

Deadlock Detection and Timeouts – A more practical approach to dealing with deadlock is deadlock detection, where the system checks if a state of deadlock actually exists.

A simple way to detect a state of deadlock is for the system to construct and maintain a ***wait-for-graph.*** One node is created in the wait-for-graph for each transaction that is currently executing. Whenever a transaction T_i is waiting to lock an item X that is currently locked by a transaction T_j, a directed edge $T_i \rightarrow T_j$ is created in the wait-for-graph. When T_j releases the lock (s) on the items that T_i was waiting for, the directed edge is dropped from the wait-for-graph. A deadlock exists in the system if and only if the wait-for-graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks the system needs to maintain the wait-for-graph, and periodically to invoke an algorithm that searches for a cycle.

Another simple scheme to deal with deadlock is the use of ***timeouts.*** This method is practical because of its low overhead and simplicity. In this method, if a transaction waits for a period longer than a system-defined timeout period, the system assumes that the transaction may be deadlocked and aborts it regardless of whether a deadlock actually exits or not.

Starvation – Another problem that may occur when we use locking is starvation. It occurs when a transaction cannot proceed for an definite period

of time while other transactions in the system continue normally. This may occur if the waiting scheme for locked items is unfair, giving priority to some transactions over others. One solution for starvation is to have a fair waiting scheme, such as using a first-come-first-serve queue. Transactions are enabled to lock an item in the order in which they originally requested the lock. Another scheme allows some transactions to have priority over others but increases the priority of a transaction the longer it waits, until it eventually gets the highest priority and proceeds.

Starvation can also occur because of victim selection if the algorithm selects the same transaction as victim repeatedly, so causing it to abort and never finish execution. The algorithm can use higher priorities for transactions that have been aborted multiple times to ignore this problem. The wait-die and wound-wait schemes ignore starvation.

Q.38. What do you mean by authentication ? How is it achieved ?

Ans. Authentication refers to the task of verifying the identity of a person/software connecting to a database. There are several methods used in authentication. The simplest and most common authentication scheme is a password to authenticate when a connection is opened to a database. Password-based authentication is used widely by operating systems as well as databases. However, the use of passwords has some drawbacks, especially over network. If an eavesdropper is able to sniff the data being sent over the network, he may be able to find the password as it is being sent over the network.

The more secure scheme is a challenge response system. The database system sends a challenge string to the user. The user encrypts the challenge string using a secret password as encryption key, and then returns the result. The database system verifies the authenticity of the user by decrypting the string with the same secret password, and checking the result with the original challenge string. This scheme ensures that no password travel across the network.

Public-key systems can be used for encryption in challenge-response systems. The database system encrypts a challenge string using the user's public key and sends it to the user. The user decrypts the string using his private key, and returns the result to the database system. The database system then checks the response. This scheme has the benefit of not storing the secret password in the database.

Q.39. Write short note on security constraint in DBMS.
(R.G.P.V., Nov./Dec. 2007)

Ans. The data stored in the database need protection from unauthorized access and malicious destruction or alteration in addition to the protection against accidental introduction of inconsistency that integrity constraints provide.

Among the forms of malicious access are unauthorized reading of data, unauthorized modification of data, and unauthorized destruction of data.

We may assign a user several forms of authorization on parts of the database such as

(i) Read authorization allows reading, but not modification of data.

(ii) Insert authorization allows insertion of new data, but not modification of existing data.

(iii) Update authorization allows modification, but not deletion of data.

(iv) Delete authorization allows deletion of data.

We may assign the user all, none, or a combination of these types of authorization.

Q.40. What do you mean by database security ? What are the different levels of security ?

Or

What does database security refer to ? ***(R.G.P.V., Dec. 2015)***

Ans. Database security refers to protection of the information contained in the database against unauthorized access or malicious access, modification, or destruction.

To protect the database, the security measures are provided at the following levels –

(i) Database System – Some database system users are authorized to access only a limited portion of the database. Other users may be allowed to issue queries, but may be forbidden to modify the data. It is the responsibility of the database system to ensure that these authorization restrictions are not violated.

(ii) Operating System – Weakness in operating system security may serve as a means of unauthorized access to the database.

(iii) Network – All database system allow remote access through terminals or networks, software-level security within the network software is as important as physical security, both on the Internet and in private networks.

(iv) Physical – Sites with computer systems must be physically secured against armed or surreptitious entry by intruders.

(v) Human – Users must be authorized carefully to reduce the chance of any user giving access to an intruder in exchange for a bribe or other favors.

Q.41. Define the term system privileges.

(R.G.P.V., Dec. 2010, June 2012)

Ans. System privileges control the creation and maintenance of many database objects, such as rollback segments, synonyms, tables and triggers. The ability to use the analyze command and the oracle database audit capability is governed by system privileges.

In general, there are various subcategories of system privileges that relate to each object. These categories find the scope of ability that the privilege grantee will have. The categories or classes of system privileges are – Admin functions, database access, tablespaces, users, undo segments, tables, clusters, indexes, synonyms views, sequences, database links, roles, transactions, PL/SQL, triggers, profiles etc.

Q.42. Define the term database privileges.

(R.G.P.V., Dec. 2010, June 2012)

Ans. All accesses in an oracle database require database privileges. Access to connect to the database, the objects the user is permitted to see and the objects the user is allowed to create are all controlled by privileges. Privileges govern the use of every database object and system resource. There are privileges needed to access objects, to create objects, to change data within tables, to execute stored procedures, to create users and so on. Because access to every object is governed by privileges, security in the oracle database is highly flexible in terms of which objects are available to which users.

Q.43. Define the term object privileges. (R.G.P.V., Dec. 2010, June 2012)

Ans. Object privileges govern a user's ability to manipulate database objects owned by other users in the database. Object privileges allow the owner of database objects, like tables, to administer access to those objects according to the types of access. The eight types of object privileges are – select, insert, update, delete, alter, index, references and execute.

Q.44. Write short note on mandatory access control. (R.G.P.V., Dec. 2014)

Ans. In some applications, an additional security policy is needed that classifies data and users based on security classes. This approach is known as mandatory access control. Mandatory access control is a type of database access control. It is based on systemwide policies that cannot be changed by individual users. In this mechanism each database object is assigned a security class, each user is assigned clearance for a security class, and rules are imposed on reading and writing of database objects by users. The DBMS determines whether a given user can read or write a given object based on certain rules that involve the security level of the object and the clearance of the user. These rules seek to ensure that sensitive data can never be passed on to a user without the necessary clearance. The SQL standard does not include any support for mandatory access control.

Q.45. Write short note on recovery system in DBMS.

Or

Write short note on recovery. ***(R.G.P.V., June 2016)***

Ans. Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently

in the database or the transaction has no effect whatsoever on the database or on any other transactions. The DBMS must not permit some operations of a transaction T to be applied to the database while other operations of T are not. This may happen if a transaction fails after executing some of its operations but before executing all of them.

Recovery from failures means that the database is restored to the most consistent state just before the time of failure. To do this, the system must keep information about the changes that were applied to data items by the various transactions. This information is kept ***system log.***

Informally, a typical strategy for recovery may be summarized as follows –

(i) If there is extensive damage to a wide portion of the database due to catastrophic failure, such a disk crash, the recovery method stores a past copy of the database that was backed up to archival storage and reconstructs a more current state by reapplying or redoing the operations of committed transactions from the backed up log, up to the time of failure.

(ii) When the database is not physically damaged but has become inconsistent due to noncatastrophic failures. The strategy is to reverse any changes that caused the inconsistency by undoing some operations.

There are two main techniques for recovery from noncatastrophic transaction failures (i) deferred update and (ii) immediate update. The deferred update techniques do not physically update the database on disk until after a transaction reaches its commit point, then the updates are recorded in the database. Before reaching commit, all transaction updates are recorded in the local transaction buffers. During commit, the updates are first recorded persistently in the log and then written to the database. If a transaction fails before reaching its commit point, it will not have changed the database in any way, so UNDO is not needed. It may be necessary to REDO the effect of the operations of a commited transaction from the log, because their effect has not yet been recorded in the database. Therefore, deferred update is also known as the NO-UNDO/REDO algorithm.

In the immediate update techniques, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are recorded in the log on disk by force writing before they are applied to the database, making recovery still possible. If a transaction fails after recording some changes in the database but before reaching its commit point, the effect of its operations on the database must be undone. That is, the transaction must be rolled back. This technique, known as the UNDO/REDO algorithm, needs both operations.

Q.46. Write short note on log based recovery. ***(R.G.P.V., May 2018)***

Ans. Refer to 45.

Q.47. What is concurrency control ? What are the recovery techniques to overcome from concurrency ? ***(R.G.P.V., May 2018)***

Or

Explain concurrency control and recovery in distributed database management system. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.21 and Q.45.

Q.48. Discuss the deferred update technique of recovery. What are the benefit and disadvantages of this technique ?

Ans. Deferred database modification technique ensures transaction atomicity by recording all database modifications in the log. In this technique, all the write statements of the transaction are applied on the database only when the transaction is partially committed. A transaction is said to be partially commited once the final action of the transaction has been executed.

When a transaction partially commits, then the information on the log associated with the transaction is used in executing the deferred writes. If the system crashes before the transaction completes its execution or if the transaction aborts, then the information on the log is ignored.

The execution of transaction T_i proceeds as follows –

Before T_i starts its execution, a record $<T_i$ **start**$>$ is written in the log. A **write**(X) operation by T_i results in the writing of a new record to the log. Finally, when T_i partially commits, a record $<T_i$ **commit**$>$ is written to the log.

For example, a banking system have accounts of A, B and C with initial balances \$1000, \$2000 and \$700 respectively. We transfer \$50 from account A to account B through transaction T_0.

For this, we write

T_0 : **read**(A);
A := A – 50;
write(A);
read(B);
B := B + 50;
write(B);

Let transaction T_1 withdraws \$100 from account C. Then transaction T_1 can be defined as

T_1 : **read**(C);
C := C – 100;
write(C);

Also, assume that these transactions execute serially in the order T_0 followed by T_1.

Log record for these transactions will have values.

<T_0 **start**>
<T_0, A, 950>
<T_0, B, 2050>
<T_0 **commit**>
<T_1 **start**>
<T_1, C, 600>
<T_1 **commit**>

Fig. 5.22 Log Record Corresponding to T_0 and T_1

Using the log, the system can handle any failure that results in the loss of information on volatile storage. The recovery scheme uses the following recovery procedure –

redo **(T_i)** sets the value of all data items updated by transaction T_i to the new values. The set of data items updated by T_i and their respective new values can be found in the log. The redo operation must be idempotent. That is, executing it several times must be equivalent to executing it once.

After a failure, the recovery subsystem consults the log to determine which transactions need to be redone. Transaction T_i is redone if and only if the log record contains both **<T_i start> and <T_i commit>** statements.

For example, we again consider our banking example with transactions T_0 and T_1 executed one after the other in the order T_0 followed by T_1.

If system fails just after the log record for the step write(B) of transaction T_0 as shown in fig. 5.23 (a). Then during recovery no redo operation will be done as we have only <T_0 **start**> in log record but not <T_0 **commit**>.

If system crash occurs just after the log record write C as shown in fig. 5.23 (b). Then during recovery only redo (T_0) is done, as we have only <T_0 **start**> and <T_0 **commit**> in log disk. At the same time, we have <T_1 **start**> in log disk but not <T_1 **commit**> so redo (T_1) will not be done. Similarly, if crash occurs just after the log record <T_1 commit> as shown in fig. 5.23 (c). Then during recovery we will perform both redo (T_0) and redo (T_1) as we have both <T_0 **start**>, <T_0 **commit**> and <T_1 **start**>, <T_1 **commit**> in log disk.

(a)	(b)	(c)
<T_0 **start**>	<T_0 **start**>	<T_0 **start**>
<T_0, A, 950>	<T_0, A, 950>	<T_0, A, 950>
<T_0, B, 2050>	<T_0, B, 2050>	<T_0, B, 2050>
	<T_0 **commit**>	<T_0 **commit**>
	<T_1 **start**>	<T_1 **start**>
	<T_1, C, 600>	<T_1, C, 600>
		<T_1 **commit**>

Fig. 5.23 Log at Three Different Situations

Advantage – The main benefit of this method is that transaction operations never need to be redone.

Disadvantages –

(i) It limits the concurrent execution of transactions because all items remain locked until the transaction reaches its commit point.

(ii) It may require excessive buffer space to hold all updated items until the transactions commit.

Q.49. Explain immediate database modification technique of recovery.

Ans. The immediate-update technique allows database modifications to be output the database while the transaction is still in the active state. Data modifications written by active transactions are called ***uncommitted modifications***.

When a system crashes or a transaction fails, old value of the data item should be used for bringing the database into the consistent state. This can be done by undo operation. Before a transaction T_i starts its execution, the record <T_i **start**> is written to the log. During its execution, any **write**(X) operation by T_i is preceded by the writing of the appropriate new update record to the log. When T_i partially commits, the record <T_i **commit**> is written to the log.

We consider example of banking system taken earlier for transactions T_0 and T_1, such that T_0 is followed by T_1. For this, log record can be given as in fig. 5.24.

<T_0 **start**>
<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>
<T_0 **commit**>
<T_1 **start**>
<T_1, C, 700, 600>
<T_1 **commit**>

Fig. 5.24 Log Record Corresponding to T_0 and T_1

In this method for recovery, we use following two operations –

(i) Undo (T_i) – Restores the value of all data items updated by transaction T_i to the old values.

(ii) Redo (T_i) – Sets the values of all data items updated by transaction T_i to the new values. We need to undo a transaction T only when log contains the record <T **start**> but does not contain the record <T **commit**>. We need to redo a transaction T only when log contains the record <T **start**> and <T **commit**> both.

For example, we consider here three situations of log record when system crash occurs at different steps in fig. 5.25.

(a)	(b)	(c)
<T_0 **start**>	<T_0 **start**>	<T_0 **start**>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 **commit**>	<T_0 **commit**>
	<T_1 **start**>	<T_1 **start**>
	<T_1, C, 700, 800>	<T_1 , C, 700, 800>
		<T_1 **commit**>

Fig. 5.25 Log Record at 3 Different Situations

Considering the fig. 5.25 (a), if system crash occurs just after the log record for the step **write**(B) of transaction T_0 has been written to the stable storage. During the recovery, we do undo (T_0) as we have only <T_0 **start**> in log record but not <T_0 **commit**>.

If system crash occurs just after the log record for the step **write**(C) of transaction T_1 has been written to stable storage as in fig. 5.25 (b).

During recovery, we do redo(T_0) and undo(T_1) as we have both <T_0 **start**> and <T_0 **commit**> in log record. But we do not have <T_1 **commit**> with <T_1 **start**> in log record. Undo(T_1) should be done first than redo(T_0) should be done. If system crash occurs just after the log record <T_1 **commit**> has been written to the stable storage space as in fig. 5.25 (c).

During recovery both redo(T_0) and redo(T_1) are performed.

Q.50. Explain the recovery process after system failure using checkpoint. ***(R.G.P.V., June 2008, Dec. 2008)***

Or

What is meant by checkpoint ? What is its importance ? What are the steps performed during checkpoint ? ***(R.G.P.V., Dec. 2010)***

Ans. Recovery using the log records contains two major problems –

(i) The search process is time consuming, as one might conclude that recovery requires just the scanning of log as a whole for recent transactions.

(ii) Most of the transactions that according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will nevertheless cause recovery to take longer.

To reduce these types of overheads, we introduce ***checkpoints.***

During execution, the DBMS maintains the log, but periodically performs checkpoints consisting of the following actions –

(i) Temporarily halting the initiation of new transaction until all the active ones are committed or aborted.

(ii) Making a backup copy of database.

(iii) Writing all log records currently residing in primary memory to stable storage.

(iv) Appending to the end to the log a record indicating that a checkpoint has occurred, then writing it to disk storage.

In this, after a failure has occurred, the recovery scheme examines the log to determine the most recent transaction T_i that started executing before the most recent checkpoint took place. For this, we search log backward from the end of log until it finds the first <checkpoint> record; then it continues the search backward until it finds the next <T_i **start**> record. This record identifies a transaction T_i. For example, consider the set of transactions {T_0, T_1,.....,T_{100}} executed in the order of the subscripts. Suppose that the most recent checkpoint took place during the execution of transaction T_{80}. Thus, only transactions T_{80}, T_{81},....,T_{100} need to be considered during the recovery scheme. Each of them need to be redone if it has committed otherwise, it needs to be undone.

Q.51. Explain various recovery techniques during transaction in detail. ***(R.G.P.V., June 2011, Dec. 2012)***

Ans. Refer to Q.48, Q.49 and Q.50.

Q.52. What are advantages and disadvantages of shadow paging techniques ? ***(R.G.P.V., Dec. 2010)***

Ans. There are some benefits of shadow paging over log-based techniques. The overhead involved in log-record output is alleviated, and recovery from crashes is faster. Even though, the disadvantages of shadow-page technique are as follows –

(i) Commit Overhead – Multiple blocks are needed by the commit of a single transaction using shadow paging to be output – the disk address of the current page table, and the actual data blocks. Log-based schemes require to output only the log records, which fit within one block for small transactions. The reduction in the overhead involved in writing an entire page table is done by implementing the page table as a tree structure, with page table entries at the leaves.

(ii) Data Fragmentation – The faster data transfer is permitted by locality. The location of the database pages is changed when they are updated due to shadow paging. Consequently, either we must resort to more difficult, higher-overhead schemes for physical storage management or we lose the locality property of the pages.

(iii) Garbage Collection – The database pages having the old version of data altered by the transaction become inaccessible, when a transaction commits. These pages are called as garbage, because they do not contain

useless information and are not part of free space. A side effect of crashes may also create garbage. Periodically, the list of free pages is made by finding all the garbage pages. This process is known as garbage collection, and imposes extra overhead and complexity on the system.

Furthermore, shadow paging is more hard compared to logging to adapt to systems that permit many transactions to execute concurrently. Some logging is needed in such systems, even if shadow paging is utilized.

Q.53. What is meant by forward and backward recovery techniques? ***(R.G.P.V., June 2012)***

Ans. The system should be prepared to recover from global failures such as system failures. All of the transactions in progress at the time of the failure are affected by such type of failures. The important point with respect to system failure is that the contents of main memory are lost. At the time of the failure, the particular state of any transaction that was in progress is therefore no longer known. The completion of such a transaction can therefore be unsuccessful, and the transaction must be undone, i.e., rolled back when the system restarts. Moreover, at restart time, it might also be essential to redo certain transactions that did successfully complete before the crash but did not manage to receive their updates transferred from the database buffers to the physical database.

One question is obviously arisen. How does the system take decision at restart time which transactions to undo and which to redo ? The answer of this question is as follows. A checkpoint is automatically taken by the system at certain stipulated intervals, normally whenever some stipulated number of entries have been written to the log. Taking a checkpoint entails physically writing a special checkpoint record out to the physical log and physically writing the contents of the database buffers out to the physical database. A list of all transactions that were in progress at the time the checkpoint was taken is given by the checkpoint record. Consider fig. 5.26 to see how this information is used.

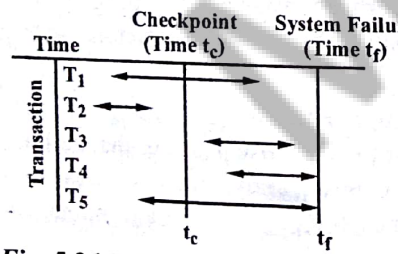


Fig. 5.26 Five Transaction Categories

It should be clear from figure that, when the system is restarted, transactions of types T_4 and T_5 must be undone, and transactions of types T_1 and T_3 must be redone. However, the transactions of type T_2 do not enter into the restart process at all, because their updates were compelled to the database at time t_c as part of the checkpoint process. The transactions that completed unsuccessfully before time t_f also do not enter into the restart process at all.

At restart time, the following procedure is followed by the system in order to identify all transactions of types T_1, T_3, T_4 and T_5.

(i) Initially, the system starts with two lists of transactions, the UNDO list and the REDO list. Set the UNDO list equal to the list of all transactions given in the most recent checkpoint record and set the REDO list is empty.

(ii) Beginning from the checkpoint record search forward through the log.

(iii) If a BEGIN TRANSACTION log entry is found for transaction T, then T is added to the UNDO list.

(iv) If a COMMIT log entry is found for transaction T, then T is moved from the UNDO list to the REDO list.

(v) The UNDO and REDO lists recognize, respectively, transactions of types T_4 and T_5 and transactions of types T_1 and T_3 when the end of the log is reached.

For undoing the transactions in the UNDO list, the system works backward through the log and for redoing the transactions in the REDO list, it works forward. Restoring the database to a consistent state by undoing work is often known as backward recovery. In the same way, restoring it to a consistent state by redoing work is often known as forward recovery.

Q.54. What is database integrity ? Explain integrity constraints. ***(R.G.P.V., June 2015)***

Or

Explain integrity concept with example. ***(R.G.P.V., June 2016)***

Ans. The term integrity refers to the accuracy or correctness of the data in the database. Integrity constraints provide a means of ensuring that changes made to the database by authorized users do not result in a loss of data consistency. So, integrity constraints guard against accidental damage to the database.

A Constraint Classification Scheme – In general, integrity constraints can be classified in four broad categories – type (domain), attribute, relation scheme, and database constraints. Basically, it can be defined as follows –

(i) Type Constraint – It specifies the legal values for a given type.

(ii) Attribute Constraint – It specifies the legal values for a given attribute.

(iii) Relation Schema Constraint – It specifies the legal values for a given relation schema.

(iv) A Database Constraint – It specifies the legal values for a given database.

Forms of Integrity Constraints – Integrity constraints have following two forms –

(i) Key Declaration – The stipulation that certain attributes form a candidate key for a given entity set. The set of legal insertions and updates is constrained to those that do not create two entities with the same value on a candidate key.

(ii) Form of a Relationship – Many-to-many, one-to-many, one-to-one. A one-to-one or one-to-many relationship restricts the set of legal relationships among entities of a collection of entity sets.

Integrity Rules – The relational model includes two integrity rules. These integrity rules implicitly or explicitly define the set of consistent database states or changes of state, or both.

(i) Entity Integrity (Integrity Rule 1) – The entity integrity designed to assure that every relation has a primary key, and that the data values for that primary key are all valid, it guarantees that every primary key attribute is not null.

(ii) Referential Integrity (Integrity Rule 2) – This rule is related to the concept of foreign key, which is used to associate two relations, with having attributes of one relation having domains that are those of the primary key of another relation. By this, we wish to ensure that a value that appears in one relation for a given set of attributes, also appears for a certain set of attributes in another relation. This condition is called ***referential integrity.***

Some Other Rules –

(i) Domain Constraints – Domain constraints specify that a domain of possible values should be associated with every attribute. Domain constraints are the most elementary form for integrity constraint. It is possible for several attributes to have the same domain, while others will be distinctly different.

For example, the attributes student_name and teachers_name might have the same domain, the set of all person names.

However the domains of salary and subject certainly ought to be distinct.

(ii) Attribute Rule – This rule says that, every attribute in every relation must be associated with a well defined domain. It must be clearly indicated about what all the possible values that the particular attribute can possess i.e., its type, its range etc.

For example, consider a relation employee (emp_name, salary).

Here, emp_name attribute must be associated with the set of all the possible names in the relation. So, we have domain 'NAME' for that. Similarly, we must associate salary with the set of valid salaries. Obviously, the salary cannot be negative. It should be number, for example, 4000, 10000, etc.

(iii) Relation Rule – These rules are concerned with the constraint over the relation. These are also known as relation constraint or relation level constraint.

For example, as in above relation employee (emp_name, salary).

We may have constraint involving single attribute would be, constraint salary > 0.

Some other integrity constraints are also use as per the requirement i.e., assertions and triggers.

(iv) Triggers – A trigger is a statement that is executed automatically by the system, as a side effect of a modification to the database. To design a trigger mechanism, we must meet two requirements –

(a) Specify the conditions under which the trigger is to be executed.

(b) Specify the actions to be taken when the trigger executes.

Q.55. Write short note on security and integrity constraints ?
(R.G.P.V., Dec. 2016)

Ans. **Security** – Security means protecting the data against accidental or intentional use by unauthorized users. The security and protection of sensitive information are very important issue. This is handled by a DBMS easily and efficiently. Permissions and access rights can be defined for one user or a group of users so that any unauthorised user cannot update or even read sensitive information.

Oracle makes several levels of security available to the DBA –

(i) Account security for a validation of users

(ii) Access security for database objects

(iii) System-level security for managing global privileges.

Account Security – If you want to access data in an Oracle database, you must have access to an account in that database. This access can be direct through user connections into a database, or indirect. Indirect connections include access through preset authorizations within database links. Each account must have a password associated with it. A database account can also be tied to an operating system account. Passwords are set for a user when user's account is created and may be altered after the account is created. A user's ability to alter the account password will be limited by the tools to which he or she is granted access. The database stores an encrypted version of the password in a data dictionary table. If the account is directly related to an operating system account, it is possible to bypass the password check and rely on the operating system authentication instead.

Object Privileges – Access to objects within a database is enabled through privileges. These allow specific database commands to be used against specific database objects through the *grant* command. For example, if the user KRISHNA owns a table called EMPLOYEE, and executes the command.

grant select on EMPLOYEE to PUBLIC;

Then all users (PUBLIC) will be able to select records from KRISHNA'S EMPLOYEE table. You can create *roles*, named groups of privileges, to simplify the administration of privileges. For applications with large numbers of users, roles greatly reduce the number of *grant* commands needed. Since *roles* can be password protected and can be dynamically enabled or disabled, they add an additional layer of security to the database.

System-level Roles and Privileges – You can use roles to manage the system-level commands available to users. These commands are *create table* and *alter index*. Actions against each type of database object are authorized through separate privileges. For instance, a user may be granted the CREATE TABLE privilege or CREATE INDEX privilege.

Integrity Constraints – Refer to Q.54.

INTRODUCTION TO – DISTRIBUTED DATABASES, BASIC CONCEPTS OF OBJECT ORIENTED DATABASE SYSTEM

Q.56. What is distributed database system ? How is it differ from the centralized database system ? ***(R.G.P.V., June 2015)***

Ans. A distributed database system is a collection of sites, connected together through some kind of communication network, in which

(i) Each site is a full database system in its own right, but

(ii) The sites have agreed to work together so that a user at any site can access data anywhere in the network exactly as if the data were all stored at the user's own site.

Distributed database is a kind of virtual database, whose component parts are physically stored in a number of distinct real databases at a number of distinct sites.

The main difference between centralized and distributed database systems is that, in the former, data reside in one single location, whereas in the latter, the data reside in several locations. There is big distinction between distributed database system and decentralized databases. A decentralized database is also stored on computers at multiple locations; however, the computers are not inter-connected by any sort of network, so that users at the various sites cannot share data. Thus, a decentralized database can be said as the collection of independent databases, rather than having the distribution of single database, at different locations.

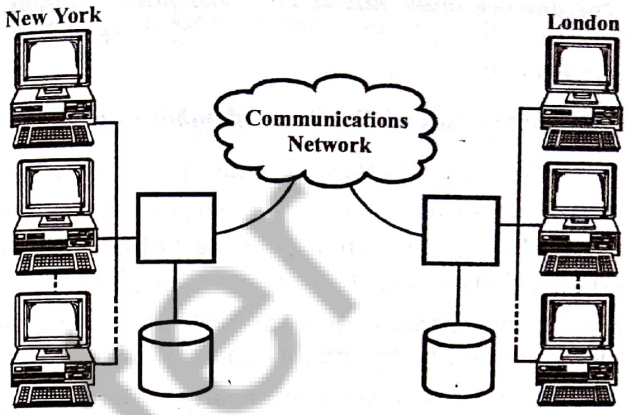


Fig. 5.27 A Typical Distributed Database System

Q.57. Explain distributed databases. Discuss various locking protocols in distributed database. ***(R.G.P.V., June 2013)***

Ans. **Distributed Databases** – Refer to Q.56.

Locking Protocols – Refer to Q.25 and Q.26.

Q.58. Write short note on distributed databases. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.56.

Q.59. What are the main objectives and the problems of distributed systems ?

Ans. **Main Objectives of Distributed Database System** – The main objectives of distributed database system are as follows –

(i) Local autonomy
(ii) No reliance on central site
(iii) Continuous operation
(iv) Location independence
(v) Fragmentation independence
(vi) Replication independence
(vii) Distributed query processing
(viii) Distributed transaction management
(ix) Hardware independence
(x) Operating system independence
(xi) Network independence
(xii) DBMS independence.

Problems of Distributed Systems – An overriding objective in distributed systems is to minimize network utilization i.e., to minimize the number and volume of messages. This objective in turn, gives rise to problems in a number of subsidiary areas, the following among them –

(i) Query processing
(ii) Catalog management
(iii) Update propagation
(iv) Recovery control
(v) Concurrency control.

Q.60. Write any two main objectives of distributed system. ***(R.G.P.V., June 2016)***

Ans. Refer to Q.59.

Q.61. What are the uses of distributed database system ?

Ans. Uses of distributed database systems are as follows –

(i) Most organizations have branches or divisions in different locations. For a given location, there may be a set of data that is required by that location only, and in addition, may require some global information.

(ii) Allowing each site to store and maintain its own database, allows immediate and access to data that are used most frequently.

(iii) Local databases can more nearly reflect an organization's administrative structure, and thereby better service its managers' needs.

(iv) If any DBMS fails, other parts of DBMS can work independently thus avoids full system shutdown at once.

Q.62. What is distributed system ? How is it differ from the centralized database system ? Give the use of distributed system. (R.G.P.V., June 2016)

Ans. Refer to Q.56 and Q.61.

Q.63. Discuss the advantages of distributed databases.

Ans. The advantages of distributed databases are as follows –

(i) Sharing – Users at a given site are able to access data stored at other sites and at the same time retain control over the data at their own site.

(ii) Availability and Reliability – Even when a portion of system (i.e., a local site) is down, the system remains available. With replicated data, the failure of one site still allows access to the replicated copy of the data from another site. The remaining sites continue to function. The greater accessibility enhances the reliability of the system.

(iii) Incremental Growth – As the organization grows, new sites can be added with little or no upheavel.

(iv) Parallel Evaluation – A query involving data from several sites can be subdivided into subqueries and the parts evaluated in parallel.

(v) Improved Performance – A distributed DBMS fragments the database by keeping the data closer where it is needed most. Data localization reduces the contention for CPU and I/O services and simultaneously reduces access delays involved in wide area networks. When a large database is distributed over multiple sites, smaller databases exist at each site. As a result, local queries and transactions accessing data at a single site have better performance because of the smaller local databases.

Q.64. Discuss the relative advantages of centralized and distributed databases. ***(R.G.P.V., Dec. 2009, 2016)***

Or

Explain relative advantages of centralized and distributed databases. ***(R.G.P.V., June 2013)***

Ans. **Advantages of Centralized Database** – The advantages of centralized databases are as follows –

(i) Centralized control
(ii) Data independence allows dynamic changes and growth potential
(iii) Data duplication eliminated with controlled redundancy
(iv) Data quality enhanced
(v) Security enforcement possible.

Advantages of Distributed Database – Refer to Q.63.

Q.65. What are the disadvantages of distributed databases ?

Ans. The disadvantages of distributed databases are its cost and complexity. A distributed system which hides its distributed nature from the end user, is more complex than the centralized DBMS. Increased complexity means that the acquisition and maintenance costs of the system are higher than those for a centralized DBMS. The parallel nature of the system means that errors are harder to avoid and those in the applications are difficult to pinpoint. In addition, the distributed system, by its nature entails a large communication overhead in coordinating messages between sites. These messages not only clutter up the network, but also degrade the system's performance. Out-of-sequence delivery or nondelivery of messages creates problems such as phantom deadlocks and blocked transactions.

Q.66. What is meant by distributed DBMS and discuss the need of distributed DBMS ? ***(R.G.P.V., June 2009)***

Ans. **Distributed DBMS** – A distributed database management system (DDBMS) is a software system that manages a distributed database while making the distribution transparent to the user.

Need of Distributed DBMS – Enterprises are usually distributed already, at least logically (into divisions, departments, workgroups, etc.), and very likely physically too (into plants, factories, laboratories, etc.) – from which if follows that the data is usually distributed already as well, because each organizational unit within the enterprise will maintain data that is relevant to its own operation. The total information asset of the enterprise is thus splintered into what are sometimes called islands of information. Distributed system provides the necessary bridges to connect those islands together. In other words, it enables the structure of the database to mirror the structure of the

enterprise – local data can be kept locally, where it most logically belongs – while at the same time remote data can be accessed when necessary.

Q.67. What is heterogeneous distributed database ? (R.G.P.V., Dec. 2015)

Ans. In a heterogeneous distributed database, defined as different sites can use different schemas, database management system software. The sites may not be aware of one another, and they can provide only limited facilities for cooperation in transaction processing. The differences in schemas are often a major problem for query processing, whereas the divergence in software becomes a hindrance for processing transactions that access multiple sites.

Q.68. Write short note on distributed processing. (R.G.P.V., June 2007)

Ans. The term distributed processing means that distinct machines can be connected together into a communications network such as the Internet, such that a single data-processing task can span several machines in the network. Communication among the various machines is handled by some kind of network management software.

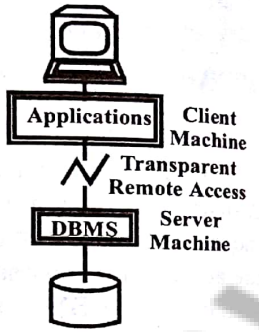


Fig. 5.28 Client(s) and Server Running on Different Machines

Some levels or varieties of distributed processing are possible. One simple case involves running the DBMS backend (the server) on one machine and the application frontends (the clients) on another. Refer to fig. 5.28.

Indeed, there are many arguments in favour of such a scheme –

(i) The first is basically just the usual parallel-processing argument – namely, many processing units are now being applied to the overall task, and server (database) and client (application) processing are being done in parallel. Response time and throughput should thus be improved.

(ii) Furthermore, the server machine might be a custom-built machine that is tailored to the DBMS (a database machine) and might thus provide better DBMS performance.

(iii) Likewise, the client machine might be a personal workstation, tailored to the needs of the end user and thus able to provide better interfaces, high availability, faster responses, and overall improved ease of use to the user.

(iv) Several different client machines might be able to access the same server machine. Thus a single database might be shared across several distinct client systems (see fig. 5.29).

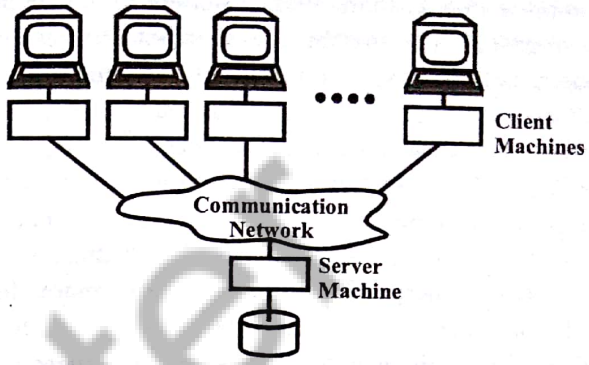


Fig. 5.29 One Server Machine, Many Client Machines

In addition to the foregoing arguments, there is also the point that running the client(s) and server on separate machines matches the way enterprises actually operate. It is common for a single enterprise – for example, a bank – to operate many computers, such that the data for one portion of the enterprise is stored on one computer and the data for another portion is stored on another. It is also common for users on one computer to need at least occasional access to data stored on another. To pursue the banking example for a moment, it is very likely that users at one branch office will occasionally need access to data stored at another. Note that the client machines might have stored data of their own, and the server machine might have applications of its own. In general, therefore, each machine will act as a server for some users and a client for others (see fig. 5.30).

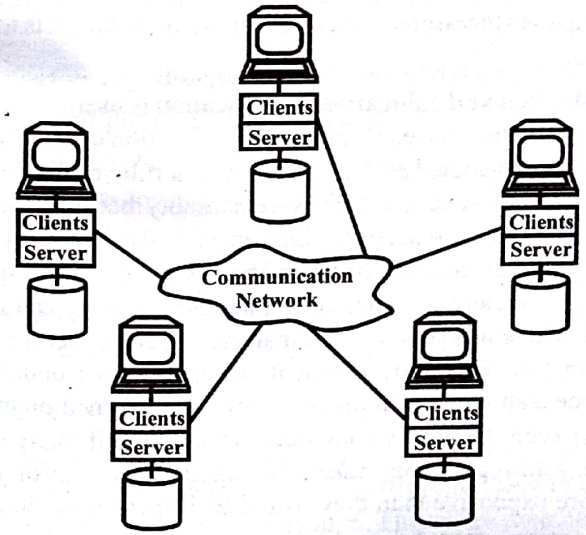


Fig. 5.30 Each Machine Runs Both Client(s) and Server

The final point is that a single client machine might be able to access different server machines. This capability is desirable because enterprises do typically operate in such a manner that the totality of their data is not stored on one single machine but rather is spread across many distinct machines, and applications will sometimes need the ability to access data from more than one machine. Basically, such access can be provided in two different ways –

(i) A given client might be able to access any number of servers, but only one at a time. In such a system it is not possible, within a single request, to combine data from two or more different servers. Furthermore, the user in such a system has to know which particular machine holds which pieces of data.

(ii) The client might be able to access many servers simultaneously. In this case, the servers look to the client-from a logical point of view – as if they were really a single server and the user does not have to know which machines hold which pieces of data.

Q.69. Explain data fragmentation, replication and allocation techniques for distributed database design. ***(R.G.P.V., Nov./Dec. 2007, June 2014)***

Ans. **Data Fragmentation** – A database is broken up into logical units, called fragments, which may be assigned for storage at the various sites.

Fragmentation is desirable for performance reasons – data can be stored at the location where it is most frequently used, so that most operations are local and network traffic is reduced.

Before we decide on how to distribute the data, we must determine the logical units of the database that are to be distributed. The simplest logical units are the relations themselves; that is, each whole relation is to be stored at a particular site.

Data Replication and Allocation – Replication is useful in improving the availability of data. The most extreme case is replication of the whole database at every site in the distributed system, so creating a fully replicated distributed database. This can improve availability remarkably because the system can continue to operate as long as at least one site is up. It also improves performance of retrieval for global queries, because the result of such a query can be obtained locally from any one site; hence, a retrieval query can be processed at the local site where it is submitted, if that site includes a server module. The disadvantage of full replication is that it can slow down update operations drastically, since a single logical update must be performed on every copy of the database to keep the copies consistent. This is true if many copies of the database exist. Full replication makes the concurrency control and recovery techniques more expensive than they would be if there were no replication.

The other extreme from full replication involves having no replication i.e., each fragment is stored at exactly one site. In this case all fragments must be disjoint except for the repetition of primary keys among vertical (or mixed) fragments. This is also known as ***nonredundant allocation***.

Between these two extremes, there is a wide spectrum of partial replication of the data i.e., some fragments of the database may be replicated while others may not. The number of copies of each fragment can range from one up to the total number of sites in the distributed system. A special case of partial replication is occurring heavily in applications where mobile workers – such as sales forces, financial planners, and claims adjustors – carry partially replicated databases with them on laptops and personal digital assistants and synchronize them periodically with the server database.

Every fragment or every copy of a fragment must be assigned to a particular site in the distributed system. This process is called data distribution or data allocation. The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site. For example, if high availability is required and transactions can be submitted at any site and if most transactions are retrieval only, a fully replicated database is a good choice. However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only. Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication. Finding an optimal or even a good solution to distributed data allocation is a complex optimization problem.

Q.70. Illustrate the issues to implement distributed databases. ***(R.G.P.V., June 2010)***

Ans. The design of a distributed database introduces three issues –

(i) How to partition the database into fragments.

(ii) Which fragments to replicate.

(iii) Where to locate those fragments and replicas.

Data fragmentation and data replication deal with the first two issues, and data allocation deals with the third issue.

Data Fragmentation – Refer to Q.69.

Data Replication and Allocation – Refer to Q.69.

Q.71. Explain different types of data fragmentation.

Ans. If relation r is fragmented, r is divided into a number of fragments $r_1, r_2, \ldots, r_n$. The fragments contain sufficient information to allow reconstruction of the original relation r. There are two techniques for fragmenting a relation – horizontal fragmentation and vertical fragmentation.

(i) Horizontal Fragmentation – In horizontal fragmentation, each fragment consists of a subset of rows of the original relation, i.e. it splits the relation by assigning each tuple of r to one or more fragments.

(ii) Vertical Fragmentation – In vertical fragmentation, each fragment consists of a subset of columns of the original relation.

Horizontal and vertical fragments are shown in fig. 5.31.

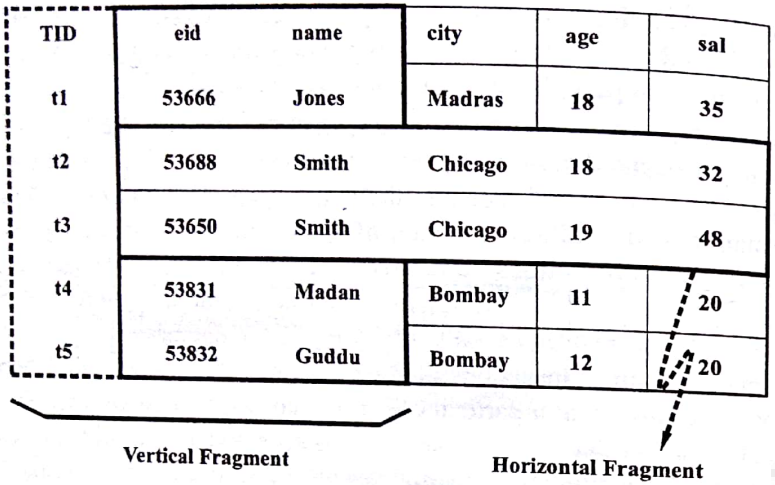


Fig. 5.31 Horizontal and Vertical Fragmentation

Typically, the tuples that belong to a given horizontal fragment are identified by a selection query. For example, employee tuples might be organized into fragments by city, with all employees in a given city assigned to the same fragment. The horizontal fragment shown in fig. 5.31 corresponds to Chicago. By storing fragments in the database site at the corresponding city, we achieve locality of reference – Chicago data is most likely to be updated and queried from Chicago, and storing this data in Chicago makes it local for most queries. Similarly, the tuples in a given vertical fragment are identified by a projection query. The vertical fragment in fig. 5.31 results from projection on the first two columns of the employees relation.

When a relation is fragmented, we must be able to recover the original relation from the fragments –

Horizontal Fragmentation – The union of the horizontal fragments must be equal to the original relation. Fragments are usually also required to be disjoint.

Vertical Fragmentation – The collection of vertical fragments should be a lossless-join decomposition.

To ensure that a vertical fragmentation is lossless-join, systems often assign a unique tuple id to each tuple in the original relation, as shown in fig. 5.31, and attach this id to the projection of the tuple in each fragment. If we think of the original relation as containing an additional tuple-id field that is a key, this field is added to each vertical fragment. Such a decomposition is guaranteed to be lossless-join.

In general, a relation can be (horizontally or vertically) fragmented, and each resulting fragment can be further fragmented.

Mixed (Hybrid) Fragmentation – We can intermix the two types of fragmentation yielding a ***mixed fragmentation.*** For example, we may combine the horizontal and vertical fragmentations of the employee relation in fig. 5.31 *into* a mixed fragmentation that includes two fragments. In this case the original relation can be reconstructed by applying UNION and OUTER JOIN (or OUTER JOIN) operations in the appropriate order.

Q.72. Compare different strategies of data allocation. Explain mixed fragmentation with examples. ***(R.G.P.V., June 2009)***

Ans. Data allocation describes the process of deciding where to locate data. Data allocation strategies are as follows –

(i) Centralized Data Allocation – In this strategy, the entire database is stored at one site.

(ii) Partitioned Data Allocation – In this strategy, the database is divided into several disjoint parts (fragments) and stored at several sites.

(iii) Replicated Data Allocation – In this strategy, copies of one or more database fragments are stored at several sites.

Mixed Fragmentation – Refer to Q.71.

Q.73. Explain how the following differ –
fragmentation transparency, replication transparency and location transparency. ***(R.G.P.V., Dec. 2009)***

Or

Explain the transparency features of a DDBMS. (R.G.P.V., Dec. 2011)

Ans. The user of a distributed database system should not be required to know either where the data are physically located or how the data can be accessed at the specific local site. This characteristic is called data transparency. The following types of transparencies are possible –

(i) Distributed or Network Transparency – This refers to freedom for the user from the operational details of the network. It may be divided into location transparency and naming transparency. Location transparency refer to the fact that the command used to perform a task is independent of the

location of data and location of the system where the command was issued. Naming transparency implies that once a name is specified, the named objects can be accessed unambiguously without additional specification.

(ii) Replication Transparency – Users view each data object as logically unique. The distributed system may replicate an object to increase either system performance or data availability. Replication allow for increased local data availability. The advantage of local data availability to access off a non update type is cheaper and even of access to a remote site not possible, access to local data is still available. Users do not have to be concerned with what data objects have been replicated, or where replicas have been placed. The major disadvantage of data replication is that the cost and complexity of updates increases, because all copies are required to be consistent with one another.

(iii) Fragmentation Transparency – A distributed database system insulates the user from knowledge of data fragmentation. The characteristic of distributed database systems is called fragmentation transparency. However from our discussion on networks and data distribution, we see that it may not always be possible to access all the data when communication link and not failures occur. The user may sense that some data is unavailable and consequently realize that data is partitioned.

To achieve locality of reference and reduced communication and redundancy costs, data is often fragmented. Fragmentation allow a subset of the relation's attributes or the subset of the relations tuples to be defined at a given side to satisfy local applications.

Q.74. List and discuss some of the factors that influenced the evaluation of the DDBMS. ***(R.G.P.V., Dec. 2011, 2013)***

Ans. The factors that influenced the evaluation of DDBMS are described as follows –

(i) Business operations which become more decentralized geographically influenced the evaluation of DDBMS.

(ii) Market needs and customer demands forced to evaluate a decentralized management style.

(iii) Another factor was the competition which increased at the global level.

(iv) The need of protecting investments in centralized DBMS software and the large number of applications based on DBMSs made the idea of data sharing attractive.

(v) Low-cost microcomputers with mainframe-like power and increasingly fast networks to connect them were created by rapid technological changes. As a result, local area networks (LANs) were adopted by companies as the basis for their computerized solutions. The increasing cost of mainframe solutions also aided the LAN's rise to prominence.

Q.75. Discuss the factor that does not appear in centralized systems that affect concurrency control and recovery in distributed system. ***(R.G.P.V., Dec. 2008)***

Ans. For concurrrency control and recovery purposes, numerous problems or factors arise in a distributed DBMS environment that are not encountered in a centralized DBMS environment. These are as follows –

(i) Dealing with Multiple Copies of the Data Items – The concurrency control method is responsible for maintaining consistency among these copies. The recovery method is responsible for making a copy consistent with other copies if the site on which the copy is stored fails and recovers later.

(ii) Failure of Individual Sites – The DDBMS should continue to operate with its running sites, if possible, when one or more individual sites fail. When a site recovers, its local database must be brought up to date with the rest of the sites before it joins the system.

(iii) Failure of Communication Links – The system must be able to deal with failure of one or more of the communication links that connect the sites. An extreme case of this problem is that ***network partitioning*** may occur. This breaks up the sites into two or more partitions, where the sites within each partition can communicate only with one another and not with sites in other partitions.

(iv) Distributed Commit – Problems arise with committing a transaction that is accessing databases stored on multiple sites if some sites fail during the commit process. The two-phase commit protocol is used to deal with this problem.

(v) Distributed Deadlock – Deadlock may occur among several sites, so techniques for dealing with deadlocks must be extended to take this into account.

Q.76. Write short note on object-orient database design. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. Object-oriented design lies in the life cycle of software system. Object-oriented design methods have evolved from object-oriented programming. The essence of OOP lies in the way that data and functions that manipulate them are brought together into a single entity called an ***object.*** An object is a data structure together with descriptions of the way it can be manipulated. Data are manipulated by one object by sending a message to

another object which contain them. The message includes identification of manipulation required and then object determines how to manipulate itself depending on the content of the message.

The term object in object-oriented design terminology is reserved to mean occurrences of a class and as such is a program run time, rather than a design consideration. So, object-oriented design deals with classes of things rather than occurrences of them. Classes in object-oriented design can be represented by abstract data types. An abstract data type describes a class in terms of operations that can be carried out on the underlying data it represents together with the properties of these operations which a user of them needs to know. It exploits the principle of information hiding. This means that in a well designed system, a module is seen by other modules only through the interface provided by that module.

The manner in which the underlying data is structured and actual mechanisms to manipulate it, are not visible in abstract data type. All that is 'on view' is the name of the class it represents and information about the interfaces it provides to other modules. So, an object-oriented design can be thought as a collection of abstract data types structured in some way in relationship to each other.

Q.77. Write short note on object-oriented database system.
(R.G.P.V., Dec. 2006)

Or

Discuss the concept of object-oriented database system.
(R.G.P.V., Dec. 2015)

Ans. Objects are mainly focused in the object-oriented systems. So, the question arises that what is object ? Then, we can say "everything is an object".

Some objects are immutable (examples might be integers i.e., 3, 42 and character strings i.e., "Rahul", "Ronal"). Other objects are mutable (examples might be the department and employee objects). Every object has a type (the object term is class).

Objects include properties and methods that operate on it. Object terminology can be shown as in fig. 5.32.

Objects are encapsulated i.e., the internal structure of such an object, say a DEPT ("department") object

Object Term	Traditional Term
immutable object	value
mutable object	variable
object class	type
method	operator
message	operator invocation

Fig. 5.32 Object Terminology

is not visible to users of that object, instead, users know only that the object is capable of executing certain operations (methods). For example, the methods that apply to DEPT objects might be

HIRE_EMP, FIRE_EMP etc.

Object Identity – Objects in an object-oriented database usually correspond to an entity in the enterprise being modeled by the database. An entity retains its identity even if some of its properties change over time.

Likewise, an object retains its identity even if some or all of the values of variables or definitions of methods change over time.

There are several forms of object identities –

(i) Value – A data value is used for identity. This form of identity is used in relational systems. For example, the primary key value of a tuple identifies the tuple.

(ii) Name – A user supplied is used for identity.

(iii) Built-in – A notion of identity is built into the data model or the programming language, and no user supplied identifier is needed.

Object Properties – Object-oriented system has some properties, few of them are as follows –

(i) Abstraction (data : = hiding) (ii) Inheritance
(iii) Encapsulation (iv) Security
(v) Reusability etc.

Q.78. Give basic concepts of object-oriented database system and design.
(R.G.P.V., June 2015)

Ans. Refer to Q.77 and Q.76.

Q.79. What are the advantages and disadvantages of object-oriented database ? ***(R.G.P.V., June 2009, Dec. 2010, June 2012)***

Ans. The advantages of OODBMS are as follows –

(i) The object approach allows modifications that are localized to a given level of an object hierarchy.

(ii) The messages to which an object responds are encapsulated along with the properties of the object. This allows constraints of various complex forms to be easily enforced.

(iii) The inheritance mechanism allows compact codes and the overriding features allow localization of changes.

The disadvantages of OODBMS are as follows –

(i) Unlike the relational approach, which started out with a formal theory and a framework for a query language, there is no formal or accepted framework of OODBMS.

MSInvites

(ii) There is no means of showing relationships among a number of objects, since each object is a self-contained unit.

(iii) Performance will likely be a problem. Techniques such as associative access and architecture features such as tagged architectures have to be investigated.

(iv) In an OODBMS, the user must know what each object class is, as well as its methods, messages, and responses. This may be the biggest stumbling block in the use of the object approach unless an intelligent user interface is provided with the database.

Q.80. What are object oriented databases ? What are its advantages and disadvantages ? ***(R.G.P.V., Dec. 2016)***

Ans. **Object-oriented Database** – Refer to Q.77.

Advantages and Disadvantages – Refer to Q.79.

Q.81. State comparison of RDBMS, OODBMS and ORDBMS. ***(R.G.P.V., Dec. 2017)***

Ans. An ORDBMS is a relational DBMS with the extensions. While an OODBMS is a programming lanaguage with a type system and permits any data object to be persistent, i.e. to survive across different program executions.

(i) RDBMS Versus ORDBMS – An RDBMS does not support the extensions. The resulting simplicity of the data model makes it. More simpler to optimize queries for efficient execution, for instance. Due to there are fewer features to master, a relational system is also more simpler to employ. Alternatively, it is less versatile as compare to an ORDBMS.

(ii) OODBMS Versus ORDBMS –

Similarities – User-defined ADTs, structured types, object identify and reference types, and inheritance are supported by both OODBMSs and ORDBMSs.

For manipulating collection types, a query language is supported by both OODBMSs and ORDBMSs. An extended form of SQL is supported by ORDBMSs and ODL/OQL is supported by OODBMSs. The similarities are by no means accidental a ORDBMSs consciously try to attach OODBMS features to an RDBMS and OODBMSS in turn have developed query languages based on relational query languages. DBMS functionality like concurrency control and recovery are given by both OODBMSs and ORDBMSs.

Differences – (a) The goal of OODBMSs is to get seamless integration with a programming language like C++, Java, Smalltalk. For ORDBMs, this type of integration is not an essential goal: SQL-92 permits us to embed SQL commands in a host language, but the interface is very clear to the SQL programmer.

(b) OODBMSs try to combine DBMS functionality to a programming language while ORDBMSs try to combine richer data types to a relational DBMS.

(c) An OODBMS is targeted at applications in which an object centric view point is suitable. It means that user sessions made up of the typical retrieving a some objects and processing on them for long periods, with associated objects fetched occasionally. Objects can be extremely large. Objects may have to be fetched in parts. Hence, attention must be paid to buffering parts of objects. It is expected that multiple applications may cache the objects they need in memory, once the objects are retrieved from disk. Hence, considerable attention is paid to making references to in-memory objects efficient. Transactions are likely to be of very long duration. Transactions holding locks until the transaction end can guide to poor performance. Hence, alternatives to two-phase locking should be employed. An ORDBMS is optimized for applications where large data collections are the focus, even though objects can have rich structure and be fairly large. It is expected that applications will retrieve data from disk extensively and optimizing disk access is still the main concern for efficient execution. Transactions are considered to be relatively short and traditional RDBMS methods are employed for concurrency control and recovery.

(e) In most OODBMSs, the query facilities of OQL are not supported efficiently, while the query facilities are the centerpiece of an ORDBMS. This condition is the result of several concentrations of effort in the development of these systems to some extent. To a significant extent, it is also a consequence of the systems being optimized for very distinct types of uses.

Q.82. Explain, why ambiguity potentially exists in multiple inheritance. Illustrate with example. ***(R.G.P.V., June 2013)***

Ans. As an example of multiple inheritance, suppose a university database, where a person can be a teacher or a student. This database may contain classes *student* and *teacher*, which are associated to the superclass *person,* to model this situation. In this section, one situation may also arise, there is also a category of students who also work as teaching assistants. A class *teaching-assistant* is created as a subclass of *student* as well as of *teacher* to model this situation.

When we use multiple inheritance, there is potential ambiguity if the same variable or method can be inherited from more than one superclass. For example, the *teacher* class may have a variable *dept* recognizing a teacher's department, and the *student* class may correspondingly have a variable *dept* recognizing a student's department. Both these definitions of the variable *dept* are inherited by the class *teaching-assistant.* Thus, in the context of teaching assistants, the use of variable *dept* becomes ambiguous.

Q.83. Explain how the concept of object identity in the object-oriented model differs from the concept of tuple equality in the relational model. ***(R.G.P.V., Dec. 2008, June 2014)***

Or

How does an object in object oriented model differ from the concept of an entity in entity relationship model. Explain. ***(R.G.P.V., June 2013)***

Ans. Objects in an object-oriented database correspond to an entity in the enterprise being modeled by the database. An entity retain its identity even if some of its properties change over time. In the same way, an object retains its identity even if some or all of the values of variables or definitions of methods change over time. This concept of identity does not apply to tuples of a relational databases. In relational systems, the tuples of a relation are distinguished only by the values that they contain.

Object identity is a stronger notion of identity than that found in programming languages or in data models not based on object-orientation. There are several forms of identity.

(i) Value – A data value is used for identity. This form of identity is used in relational systems. For example, the primary key value of a tuple identifies the tuple.

(ii) Name – A user-supplied name is used for identity. This form of identity is used for files in file system. Each file is given a unique name to identify it.

(iii) Built-in – A notion of identity is built into the data model or programming language, and no-user-supplied identifier is needed. This form of identity is used in object-oriented systems. Each object is given an identifier when that object is created automatically by the system.

The identity of objects is a conceptual notion. Actual system require a physical mechanism to identify objects. For identifying humans, names along with other information, such as date and place of birth, are often used as identifiers. Object-oriented systems use an object identifier to identify objects. Object identifiers are unique. Object identifiers are not necessarily in a form with which humans are comfortable; for example, they could be long numbers. The ability to store the identifier of an object as a field of another object is more important than having a name that is easy to remember.

As an example of the use of object identifiers, one of the attributes of a *person* object may be a *spouse* attribute, which is the identifier of the *person* object corresponding to the first person's spouse. Hence, the *person* object store a reference to the object representing the person's spouse.

The identifier is generated automatically by the system. On the other hand, in a relational database, the *spouse* field of a *person* relation will be a unique identifier of the person's spouse, generated outside the database system.

Q.84. State and explain the features of object oriented data model. Use banking application as an example. ***(R.G.P.V., June 2010)***

Ans. The main concepts of the object-oriented data model are the structure of objects and the notions of classes, inheritance, and object identity.

(i) Object Structure – An *object* represents an entity in the E-R model. The object-oriented paradigm depends on encapsulation of data and code related to an object into a single unit. The interface between an object and rest of the system is defined by a set of messages. It means that all interactions between an object and the rest of system are through messages.

Generally, an object has associated with it

(a) A set of variables containing the data for the object. Variables represent attributes in the E-R model.

(b) A set of messages to which the object responds.

(c) A set of methods, each of which contains a body of code to implement a message.

Consider *employee* entities in a bank database. The annual salary of different employees is calculated differently. For instance, tellers may get a bonus based on how many hours they have worked, in contrast managers may get a bonus based on the bank's performance. The code for calculating the salary is encapsulated with each employee as a method that is executed in response to an *annual-salary* message.

All employee objects respond to the annual-salary message differently. All employee objects represent the same interface by encapsulating within the *employee* object itself the information about how to calculate the annual salary.

There may be *read only* or *update* methods of an object. An update method may change the values of the variables, whereas a read-only does not change the values of the variables. The messages can also be classified as read-only or update based on the method that implements the message.

The derived attributes of an E-R model entity are expressed as read-only messages in the object-oriented model. For instance, the derived attribute *employment-length* of an *employee* entity is expressed as an *employment-length* message of an *employee* object. The employment length may be determined by the method implementing the message by subtracting the joining-date from the current date.

(ii) Object Classes – In the object-oriented data model, the notion of a class represents the notion of an entity set in the E-R model. The employees, customers, accounts and loans are some examples of classes in our bank database.

The definition of the class *employee* in pseudocode is shown in fig. 5.33. According to this definition, each object of the class *employee* has the variables *name* and *address* of type string, joining-date of type date, and salary of type integer. Each object answers to the five messages, namely annual-salary, get-address, get-name, put-address, and employment-length

For handling the messages, the methods are defined separately from the class definition. For instance, the methods *get-address* and *put-address( )* are defined by the pseudocode –

```
string get-address( ) {
    return address;
}
int put-address (string new-address) {
    address = new-address;
}
```

whereas the method employment-length( ) is defined as –

```
int employment-length( ) {
    return today( ) - joining-date;
}
```

Here, it is assumed that the today( ) function returns the current date, and the "-" operation on dates the interval between the two dates.

```
class employee {
    /* Variables */
        string name;
        string address;
        date    joining-date;
        int     salary;
    /* Messages */
        int   annual-salary( );
        string get-name( );
        string get-address( );
        int   put-address(string new-address);
        int   employment-length( );
};
```

Fig. 5.33 The Class Employee Definition

(iii) Inheritance – There are several classes in an object-oriented database schema. Even though, several classes are similar. For instance, the class of bank employees is expected to be identical to the class of bank customers, because both define variables for name, address and so on. Even though, there are variables specific to customers (e.g., credit-rating) and variables specific to employees (e.g., salary). Then, it is preferred to define a representation for the common variables in one place.

A specialization hierarchy (e.g., the "ISA" relation-ship) is required to place classes to permit the direct representations of similarities among classes in the E-R model. For instance, *employee* is a specialization of *person*. Likewise, *customer* is a specialization of *person*.

The concept of class hierarchy is same as the concept of specialization hierarchy in the E-R model. The specializations of a *person* class represent the employees and customers classes. Variables and methods specific to both classes are associated.

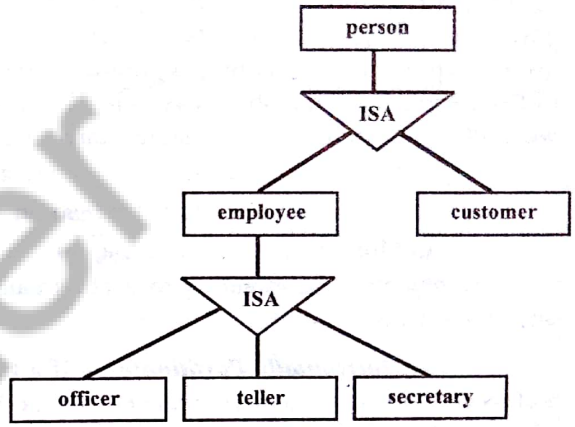


Fig. 5.34 Specialization Hierarchy for the Bank Example

An E-R diagram with a specialization hierarchy is shown in fig. 5.34 to represent people in the blank database. The corresponding class hierarchy is shown in fig. 5.35.

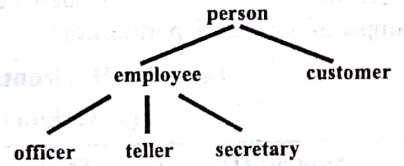


Fig. 5.35 Class Hierarchy Corresponding to Fig. 5.34

The keyword ***isa*** specifies that a class is a specialization of another class. The specializations of a class are referred to as ***subclasses.*** For instance, *customer* is a subclass of *employee*. Conversely, *person* is a superclass of *customer*.

An object representing a secretary has all the variables of class *secretary*. Moreover, a secretary object also contains all the variables of classes *employee* and *person*. This property is known as the inheritance of variables.

Substitutability is an important advantage of inheritance in object-oriented systems. Any method of a class X can be invoked with any object belonging to any subclass Y of X. This feature offers code reuse, because the messages, methods, and functions are not needed to be rewritten for objects of class Y. For example, the message get-address( ) is defined for the class person, and it can be called with a person object, or with any object belonging to any subclass of *person,* like *employee* or *secretary*.

(iv) Object Identity – Refer to Q.83.

MsinVenter

Q.85. How can we map a class to a relation ? (R.G.P.V., June 2009)

Ans. If we need to store classes (actually, objects) on a disk, there must be some way of mapping them to RDBMS structures. Because, objects are otherwise in-memory constructs, they do not have life beyond the execution cycle of a program. The moment a program is removed from the main memory of the computer, all the objects associated with that program also die. When we think of mapping objects to tables, three possibilities emerge –

(i) One object maps to exactly one table.

(ii) One object maps to more than one table.

(iii) More than one objects map to one table.

The objects corresponding to a class can be partitioned horizontally and/or vertically.

(i) Horizontally Partitioning – If a few instances (i.e., objects) of a class are referenced to more frequently than others, it may be a good idea to place the frequently-accessed instances in one physical table. The ones not frequently accessed can be put in another table. It is imperative that application using these tables should know which table contains the frequently-accessed data, of it would not benefit from such a partitioning scheme. Table 5.3 shows an example of horizontal partitioning.

Table 5.3 Horizontal Partitioning

(a) Original Table

Student ID	*Student Name*	*Marks*
01	Atul	62
02	Ana	70
03	Jui	82
04	Harsh	89

(b) Horizontal Partition Number 1

Student ID	*Student Name*	*Marks*
01	Atul	62
02	Ana	70

(c) Horizontal Partition Number 2

Student ID	*Student Name*	*Marks*
03	Jui	82
04	Harsh	89

(ii) Vertical Partitioning – There are after situations when some attributes of a class are more frequently accessed than other. This means that the class may have varying ***access patterns***. In such cases, the table corresponding to the class can be partitioned vertically. The frequently accessed attributes can be placed in one table and the other attributes can be placed in another. Table 5.4 shows an example of vertical partitioning.

Table 5.4 Vertical Partition

(a) Original Table

Student ID	*Student Name*	*Marks*
01	Atul	62
02	Ana	70
03	Jui	82
04	Harsh	89

(b) Vertical Partition Number 2

Student ID	*Student Name*
01	Atul
02	Ana
03	Jui
04	Harsh

(c) Vertical Partition Number 3

Student ID	*Marks*
01	62
02	70
03	82
04	89

Now we assume an example of mapping a simple class to a table. Let us assume that we have a class student, having two attributes, student name, and marks. This is the view of our object model. When we want to map this to an appropriate table, there are two main steps – (i) outline the corresponding table model for this class, and (ii) Write the SQL code corresponding to the table model.

While transforming the object model to the table model, we add a record identifier (student ID). Moreover, we also provide greater details about the

MSInventer

table model. For instance, we specify that the student ID must be unique and must not be null, which makes it our ideal primary key. We also mention that the student name must have some value – it cannot be null. However, because two or more students can have the same name, this cannot be a primary key but a secondary one. This would mean that we create an index on the student name, in addition to the default index on the student ID, so facilitating a quicker search based on the student name. Marks is a simple attribute without any regards to the key. The entire process is shown in fig. 5.36.

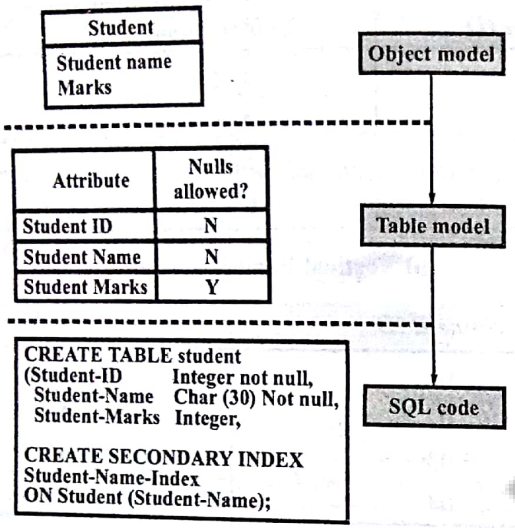


Fig. 5.36 Mapping a Class to a Table

⌘⌘⌘

RGPV

B.E. (Fourth Semester) EXAMINATION, Dec., 2010
(New Scheme)
(Information Technology Engg. Branch)
DATABASE MANAGEMENT SYSTEM
[IT–403(N)]

Note : Attempt all the *five* questions. All questions carry equal marks.

1. (a) What is meant by the aggregation ? Explain with example. 8
(See Unit-II, Page 104, Q.75)

(b) Develop an ER diagram for sales and purchase management system. 12

Or

(a) Compare file management system with database management system. 8
(See Unit-I, Page 6, Q.7)

(b) What are the different types of entities and relationship in an education system ? Draw an ER diagram. **(See Unit-II, Page 107, Prob.2)** 12

2. (a) Compare the procedure of Insert, Delete and Update operations in network and relational model. **(See Unit-I, Page 24, Q.29)** 12

(b) What is the significance of using the views ? Write a command to create the view. **(See Unit-II, Page 69, Q.37)** 8

Or

(a) Explain the different CODD rules. **(See Unit-III, Page 131, Q.16)** 8

(b) What are the Integrity rules ? Explain the concept of Intension and Extensions. **(See Unit-II, Page 49, Q.9)** 12

3. Solve the following on the given database. Write the SQL and Relational Algebra of given queries : 20

S (Sno, Sname, City, Status)
P (Pno, Pname, Color, Weight)
SP (Sno, Pno, Qty)

(i) Get supplier name who supplies at least one red part.
(ii) Get supplier number who supplies maximum quantity.
(iii) Get supplier number who supplies quantity greater then average quantity.
(iv) Increase the quantity of part P_1 by 10%.
(v) Change the color of Red parts supplied by Ajay to Green.

(See Unit-III, Page 155, Prob.9)

Or

Solve the following with pure tuple calculus and QUEL Tuple calculus.
Consider the following database :
EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)

MsinVenter

DEPT (deptno, dname, loc)

(i) Get the name of all employees working for the 'Sales' department.

(ii) Get the name, location of department of all employees whose pay is less than 5000.

(iii) Get the name of all departments who have more than 5 employees.

(iv) Get the name of all departments who have at least one employee.

(v) Decrease the salary by 100 for all employees in "Account" department.

4. (a) What is overlapping of candidate keys ? What are the problems incurred due to this overlap and how these problems are rectified ? 10

(b) Normalize the following database (empno, ename, job, sal, day, mgr, depno, dname, loc). (See Unit-IV, Page 195, Q.32) 10

Or

(a) What is meant by projection join dependency ? 10 (See Unit-IV, Page 213, Q.52)

(b) Explain the similarities and dissimilarities between BCNF and 3rd Normal form. (See Unit-IV, Page 209, Q.47) 10

5. (a) What is meant by checkpoint ? What is its importance ? What are the steps performed during checkpoint ? (See Unit-V, Page 252, Q.50) 10

(b) What are advantages and disadvantages of object oriented database mean ? (See Unit-V, Page 271, Q.79) 10

Or

(a) What are advantages and disadvantages of shadow paging techniques ? (See Unit-V, Page 253, Q.52) 8

(b) Define the following terms : 12

(i) System privileges (See Unit-V, Page 246, Q.41)

(ii) Database privileges (See Unit-V, Page 247, Q.42)

(iii) Object privileges (See Unit-V, Page 247, Q.43)

(iv) Privacy of data

RGPV

B.E. (Fourth Semester) EXAMINATION, June, 2011
(Information Technology Engg. Branch)
DATABASE MANAGEMENT SYSTEM
[IT-403(N)]

Note : Attempt all questions. Internal choice is given. All questions carry equal marks.

1. (a) Discuss the main characteristics of the database approach and how it differs from traditional file system. (See Unit-I, Page 6, Q.7) 10

(b) What is the difference between logical data independence and physical

data independence ? Which one is harder to achieve ? Why ? 10 (See Unit-I, Page 34, Q.43)

Or

2. (a) Describe the three-schema architecture. Why do we need mapping between schema levels ? How do different schema definition languages support this architecture ? (See Unit-I, Page 14, Q.13) 10

(b) Discuss the naming conventions used for ER schema diagram. 10 (See Unit-II, Page 83, Q.53)

3. (a) Discuss the main categories of data models. (See Unit-I, Page 17, Q.18) 10

(b) What is the difference between a key and a superkey ? Why do we designate one of the candidate keys of a relation to be the primary key ? (See Unit-II, Page 87, Q.59) 10

Or

4. (a) Discuss the entity integrity and referential integrity constraints. Why is each considered important ? (See Unit-II, Page 47, Q.6) 10

(b) Compare among Hierarchical, Network and Relational data model. 10 (See Unit-I, Page 25, Q.30)

5. (a) What are the relational algebra operations supported in SQL ? Write the SQL statements for each operation. (See Unit-III, Page 123, Q.7) 10

(b) What are the steps involved in query processing ? How would you estimate the cost of the query ? (See Unit-III, Page 166, Q.39) 10

Or

6. (a) What are the conditions to be fulfilled for two relations to be involved in a Union operation ? Why do the Union, Intersection and Set Difference operation require the operand relations to be Union compatible ? (See Unit-II, Page 60, Q.23) 10

(b) In a tuple relational calculus query with n tuple variables, what would be the typical minimum number of join conditions ? Why ? What is the effect of having a smaller number of join conditions ? 10

7. (a) What is normalization ? Explain the normalization techniques using functional dependencies with relevant examples. 10 (See Unit-IV, Page 187, Q.16)

(b) Discuss the purpose of Boyce Codd normal form and describe how BCNF differ from and is stronger than 3NF. Illustrate your answer with an example. (See Unit-IV, Page 204, Q.42) 10

Or

8. (a) List all FDs satisfied by the following table T. Note that these are all potentially possible functional dependencies but may not hold true for all states of this relations : 10

MSinventer

Table - T

	A	B	C	D
1	a_1	b_1	c_1	d_1
2	a_1	b_1	c_2	d_2
3	a_1	b_2	c_2	d_1
4	a_1	b_2	c_4	d_4

(b) Describe the characteristics of a table that is not in normalized form. Describe how such a table is converted to a first normal form relation. **(See Unit-IV, Page 187, Q.19)** 10

9. (a) Explain the security features provided in commercial query languages. 10

(b) Explain *four* important properties of transaction that a DBMS must ensure to maintain database. **(See Unit-V, Page 218, Q.2)** 10

Or

10. (a) Explain various recovery techniques during transaction in detail. 10 **(See Unit-V, Page 253, Q.51)**

(b) Explain the following protocols for concurrency control : 10

(i) Lock based protocols **(See Unit-V, Page 234, Q.25)**

(ii) Time stamp based protocol **(See Unit-V, Page 237, Q.31)**

RGPV

B.E. (Fourth Semester) EXAMINATION, Dec., 2011
(Information Technology Engg. Branch)
DATABASE MANAGEMENT SYSTEM
(IT–403)

Note : Attempt any *five* questions. All questions carry equal marks.

1. (a) Define the following terms – 5

(i) Data redundancy and inconsistency **(See Unit-I, Page 6, Q.6)**

(ii) Referential Integrity **(See Unit-II, Page 49, Q.8)**

(b) What is the role of a DBMS and what are its advantages ? 5 **(See Unit-I, Page 9, Q.9)**

(c) Explain how the entity relationship (ER) model helped produce a more structured relational database design environment. 10 **(See Unit-II, Page 81, Q.50)**

Or

2. (a) A university database contains information about professors (identified by social security number) and courses (identified by courseid) professors teach courses, each of the following situations concerns the teaches relationship set. For each situation draw an E-R diagram – 10

(i) Professors can teach the same course in several semesters and each offering must be recorded.

(ii) Professors can teach the same course in several semesters and only the most recent such offering needs to be recorded.

(iii) Every professor must teach some course.

(iv) Every professor teaches exactly one course.

(v) Every professor teaches exactly one course and every course must be taught by some professor.

(b) What are the major differences between a file processing system and a DBMS ? **(See Unit-I, Page 6, Q.7)** 10

3. (a) Explain the advantages of Relational Model and also explain the Data Control facilities in Relational Model. **(See Unit-I, Page 30, Q.34)** 10

(b) Write short notes on the following – 10

(i) Network data model **(See Unit-I, Page 19, Q.22)**

(ii) Hierarchical data model **(See Unit-I, Page 18, Q.20)**

Or

4. (a) Discuss the candidate key, primary key, super key, composite key and alternate key. **(See Unit-II, Page 86, Q.58)** 10

(b) Why are entity integrity and referential integrity important in database? Explain with suitable examples. **(See Unit-II, Page 47, Q.6)** 10

5. (a) Given the relation schema – 10

S_emp (Id, Last_name, First_name, Userid, Start-date, Designation, Dept_id, Salary)

Give queries expressed in SQL –

(i) Display the last name, first name and department Id for all employees who do not work in department 44.

(ii) Display the last name, user id and start date for all employees hired on March 8, 1990.

(iii) Display the last name, first name and salary of all employees whose salary is less than $ 1,200.

(iv) Display the last name, first name and salary of all the employees who are managers. Order the result in ascending order by salary.

(v) Increase the salary of all employees of department 41 by 1000. **(See Unit-III, Page 150, Prob.4)**

(b) What are the various characteristics of SQL ? Discuss five aggregate functions with a suitable example. **(See Unit-III, Page 122, Q.5)** 5

(c) Discuss the selection, projection and join operator of relation algebra with a suitable example. **(See Unit-II, Page 64, Q.32)** 5

Or

MSIT Vertical

6. (a) Employee (SSN, name, age, dno) 10
Salary (SSN, salary)
Works on (Projects#, SSN)
Project (Projects#, project_name, location)
For each of the following queries give an expression in SQL :
(i) Display the names of projects at "delhi".
(ii) Find the project name of employee whose salary is greater than 10000.
(iii) Retrieve the name and SSN of employees working on Project# A 100.

(See Unit-III, Page 159, Prob.12)

(b) Discuss the various features of relational data model in detail. 10

(See Unit-I, Page 23, Q.27)

7. (a) What do you mean by Normalization ? Explain BCNF and 3 NF with suitable example. (See Unit-IV, Page 206, Q.43) 10

(b) What do you mean by decomposition of a relation ? Consider the relational scheme R(A, B, C, D, E, F) and FDs

$A \rightarrow BC, C \rightarrow A, D \rightarrow E, F \rightarrow A, E \rightarrow D$

Is the decomposition of R into R1(A, C, D), R2(B, C, D) and R3(E, F, D) lossless ? Explain the requirements for lossless decomposition and dependency preserving. (See Unit-IV, Page 199, Q.38) 10

Or

8. (a) Define the following terms – 10
(i) Functional dependency (See Unit-IV, Page 179, Q.1)
(ii) Lossless decomposition (See Unit-IV, Page 198, Q.37)
(iii) Dependency preservation (See Unit-IV, Page 197, Q.36)
(iv) Third Normal Form (See Unit-IV, Page 192, Q.25)

(b) Describe the term MVD in the context of relational database management system by giving an example. Discuss 4 NF and 5 NF also. (See Unit-IV, Page 215, Q.54) 10

9. (a) Describe the three most common concurrent transaction execution problems. Explain how concurrency control can be used to avoid those problems. (See Unit-V, Page 233, Q.22) 10

(b) What DBMS component is responsible for concurrency control ? How is this feature used to resolve conflicts ? 10

(See Unit-V, Page 233, Q.23)

Or

10. (a) Explain the transparency features of a DDBMS. 10

(See Unit-V, Page 267, Q.73)

(b) List and discuss some of the factors that influenced the evaluation of the DDBMS. (See Unit-V, Page 268, Q.74) 10

RGPV

B.E. (Fourth Semester) EXAMINATION, June, 2012
(Grading/Non-Grading)
(Information Technology Engg. Branch)
DATABASE MANAGEMENT SYSTEM
(IT–403)

Note : Attempt any *five* questions. All questions carry equal marks.

1. (a) Explain three level architecture. What are its objectives ?

(See Unit-I, Page 15, Q.14)

(b) Develop an ER diagram for library management system.

(See Unit-II, Page 81, Q.49)

Or

(a) Explain the distinction between specialization and generalization concepts. (See Unit-II, Page 100, Q.72)

(b) What are the different types of entities and relationships in railway reservation system ?

2. (a) Explain the distinction between representation of association, relationship and basic operation in data models.

(b) What are the integrity rules ? Explain the significance of primary key and foreign key in employee database. (See Unit-II, Page 47, Q.6)

Or

(a) What is the significance of using the views ? Write a command to create the view. (See Unit-II, Page 69, Q.37)

(b) A relation consists of attributes A, B, C, D, E, F, G and H :
(i) No single attribute has unique values.
(ii) The combination of attributes A and E is unique.
(iii) The combination of attributes B and G is unique.
Select primary key for this relation and indicate any alternate keys.

3. Solve the following on the given database. Write the SQL and Relational Algebra for given Queries :
S (SNo, Sname, City, States)
P (Pno, Pname, Color, Weight)
SP (Sno, Pno, QTY)
(i) Get supplier name who supply atleast one red part.
(ii) Get supplier number who supplies maximum quantity.
(iii) Get supplier number who supply quantity greater than average quantity.

Mrinvenier

(iv) Increase the quantity of Part P_1 by 10%.

(v) Change the color of Red Parts supplied by Ajay to Green.

(See Unit-III, Page 155, Prob.9)

Or

Solve the following with pure tuple calculus and Quick tuple calculus.
Consider the following database :
EMP (empno, ename, Job, mgr, hiredate, sal, comm, deptno)
DEPT (deptno, dname, loc)

(i) Get full detail of employee in relation EMP.

(ii) Get employee number for department number 10.

(iii) Get employee names that belong deptno 40 and having salary > 1000.

(iv) Get employee number and name who are working in "Sales" department.

(v) Decrease the salary by 100 for all employee in "Account" department.

4. (a) What are the anomalies occurred due multi-value dependence and had the anomalies are rectified ?

(b) Explain the cases where 4th normal form is applicable. How does it remove the anomalies ? (See Unit-IV, Page 211, Q.49)

Or

(a) Normalize the following database :
(empno, ename, job, sal, doj, mgr, deptno, dname, loc)
(See Unit-IV, Page 195, Q.32)

(b) What is meant by projection dependency ? (See Unit-IV, Page 213, Q.52)

5. (a) What is meant by forward and backward recovery techniques ?
(See Unit-V, Page 254, Q.53)

(b) What are advantages and disadvantages of object oriented database mean ? (See Unit-V, Page 271, Q.79)

Or

Describe the following terms :

(i) System privileges (See Unit-V, Page 246, Q.41)

(ii) Database privileges (See Unit-V, Page 247, Q.42)

(iii) Object privileges (See Unit-V, Page 247, Q.43)

(iv) Privacy of data

RGPV

B.E. (Fourth Semester) EXAMINATION, Dec., 2012
DATABASE MANAGEMENT SYSTEM
(IT–403)

Note : Attempt all questions, each questions carry equal marks.

1. (a) Compare file system with database system. (See Unit-I, Page 6, Q.7)

(b) Explain the architecture of DBMS. (See Unit-I, Page 10, Q.10)

Or

2. (a) Construct an ER diagram for a car insurance company that has a set of customers, each of whom owns one/more cars. Each car has associated with it zero to any number of recorded accidents.
(See Unit-II, Page 107, Prob.1)

(b) Why is the administration of a database system required ? What are the various functions associated with database administration ?
(See Unit-I, Page 39, Q.48)

3. (a) List the possible types of relations that may exist between two entities. How would you realize that into tables for a binary relation ?
(See Unit-II, Page 97, Q.66)

(b) Define data model. Explain the different types of data models with relevant examples. (See Unit-I, Page 24, Q.28)

Or

4. (a) Explain the following :

(i) Entity and Attributes (See Unit-I, Page 39, Q.50)

(ii) Simple and composite attributes (See Unit-I, Page 40, Q.53)

(iii) Single value and multivalue attributes (See Unit-I, Page 40, Q.53)

(iv) Derived attributes (See Unit-I, Page 40, Q.53)

(b) Discuss the three level architecture of DBMS. Explain how does it lead to data independence. (See Unit-I, Page 34, Q.41)

5. (a) Consider the following relations :
Employee (empID, FirstName, LastName, address, DOB, sex, position, deptNo)
Department (dtptNo., deptName, mgr, empID)
Project (projNo, projName, deptNo)
Work_on (empID, projNo, hours worked)
Write the SQL statements for the following :

(i) List the name and addresses of all employees who work for th IT department.

(ii) List the total hours worked by each employee, arranged in orde of department number and within department, alphabetically b employee surname.

(iii) List the total number of employees in each department for thos departments with more than 10 employees.

(iv) List the project number, project name and the number employees who work on that project.

(b) Explain any four aggregate functions of SQL with an example f each. (See Unit-III, Page 120, Q.

MSInverter

Or

(a) With relevant examples discuss the following in SQL.
(i) Data Definition Language. (ii) Data Manipulation Language
(iii) Data Control Language (iv) Views
(See Unit-III, Page 127, Q.13)

(b) Given the relational schemes :
ENROL (S#, C#, Section) - S# represents students number
TEACH (Prof, C#, Section) - C# represents course number
ADVISE (Prof, S#) - Prof is thesis advisor of S#
PRE_REQ(C#, Pre_C#) - Pre_C# is prerequisite course
GRADES (S#, C#, Grade, Year)
STUDENT (S#, Sname) - Sname is student name
Give queries expressed in relational algebra and domain calculus for the following :
(i) List all students taking courses with Zeba.
(ii) List all students taking atleast one course that their advisor teaches.
(iii) List those professors who teach more than one section of the same course.

(a) With the help of an example relation and the dependency diagram, explain the concept of functional dependency.
(See Unit-IV, Page 179, Q.1)

(b) (i) What are relational algebra operations supported in SQL ? Write the SQL statement for each operation. (See Unit-III, Page 123, Q.7)
(ii) Justify the need for normalization with examples.
(See Unit-IV, Page 186, Q.15)

Or

(a) What is normalization ? Give the various normal forms of relational schema and define a relation which is in BCNF and explain with suitable example. (See Unit-IV, Page 215, Q.55)

(b) Compare BCNF versus 3NF. (See Unit-IV, Page 209, Q.47)

(a) Explain various recovery techniques during transaction in detail.
(See Unit-V, Page 253, Q.51)

(b) Describe about the testing of serializability.
(See Unit-V, Page 229, Q.18)

Or

(a) What is concurrency control ? How is it implemented in DBMS ? Explain. (See Unit-V, Page 232, Q.21)

(b) Discuss on two-phase locking protocol. (See Unit-V, Page 234, Q.25)

RGPV

B.E. (Fourth Semester) EXAMINATION, June, 2013
DATABASE MANAGEMENT SYSTEM
(IT–403)

Note : Attempt any one question from each unit. All questions carry equal marks.

Unit-I

1. (a) (i) What are the main advantages of DBMS ?
(See Unit-I, Page 8, Q.8)

(ii) What are the main functions of database administrator ?
(See Unit-I, Page 36, Q.45)

(b) (i) Differentiate between physical and logical data independence.
(See Unit-I, Page 34, Q.42)

(ii) Explain primary key, super keys, candidate keys with examples.
(See Unit-II, Page 86, Q.58)

Or

2. (a) Explain strong and weak entities. How weak entity can be converted to a strong entity. Outline what sort of redundancy will result by converting weak entity to strong entity.

(b) Explain the following constraints while designing E-R diagram.
(i) Total and partial constraints
(ii) Disjoint and partial constraints
(iii) Condition defined and user defined constraints.
(See Unit-II, Page 103, Q.74)

Unit-II

3. (a) Differentiate between hierarchical, network and relational data models by taking examples. Also discuss their advantages and disadvantages.
(See Unit-I, Page 30, Q.35)

(b) Explain with examples –
(i) Integrity constraints (ii) Relational schema
(iii) Domains (iv) Extension and intension of relation.
(See Unit-II, Page 49, Q.12)

Or

4. (a) What are the characteristics of relation. Explain domain, tuple, attributes, degree, cardinality of relation. (See Unit-II, Page 50, Q.15)

(b) Discuss the concepts of network and hierarchical model.
(See Unit-I, Page 21, Q.23)

Msinventer

Unit-III

. **(a)** Explain various types of join with examples. **(See Unit-II, Page 69, Q.36)**

(b) What are views ? Explain what are the major problems with processing update operations expressed in terms of views.

(See Unit-II, Page 69, Q.37)

Or

. Consider the relational database given below. Give expressions in QBE and for each of the following queries.

Employee (Person-name, street, city)

Works (Person-name, company-name, salary)

Company(Company-name, city)

Manager (person-name, Manager-name)

(i) Give all managers in the database a 10% raise.

(ii) Find name of all employees who work in 'xyz'.

(iii) Find all employees who do not work in 'xyz'.

(iv) Modify the database so that the 'Jones' now lives in Newtown.

Unit-IV

. **(a)** Explain functional dependency and trivial functional dependency by taking examples. List all functional dependencies satisfied by the relation of figure given below –

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_3	c_1
a_2	b_3	c_3

(See Unit-IV, Page 182, Q.8)

(b) Give closure of the following set F of functional dependencies for the relation

R = (A, B, C, D, E) where

F = {A → BC

B → D

CD → E

E → A}

Also list all candidate keys of R **(See Unit-IV, Page 185, Q.14)**

Or

(a) What are the design goals of relational databases ? Explain why each is desirable. **(See Unit-II, Page 46, Q.2)**

(b) Let R_1, R_2, R_3, ... R_n be the decomposition of schema U. Let u(U) be a relation and let $r_i = \pi R_i(u)$ show that $u \subseteq r_1 \bowtie r_2 \bowtie r_3 ..... r_n$. **(See Unit-IV, Page 196, Q.35)**

Unit-V

9. **(a)** Explain distributed databases. Discuss various locking protocols in distributed database. **(See Unit-V, Page 259, Q.57)**

(b) Explain relative advantages of centralized and distributed databases. **(See Unit-V, Page 261, Q.64)**

Or

10. **(a)** How does an object in object oriented model differ from the concept of an entity in entity relationship model. Explain. **(See Unit-V, Page 274, Q.83)**

(b) Explain, why ambiguity potentially exists in multiple inheritance. Illustrate with example. **(See Unit-V, Page 273, Q.82)**

RGPV

B.E. (Fourth Semester) EXAMINATION, Dec. 2013

DATABASE MANAGEMENT SYSTEM

(IT–403)

Note : Attempt all questions. Each question carry equal marks.

1. **(a)** Define the following terms –

(i) Data models **(See Unit-I, Page 16, Q.17)**

(ii) Domain Constraints **(See Unit-II, Page 49, Q.10)**

(b) Draw an E-R diagram for a Library Management System, make suitable assumptions. **(See Unit-II, Page 81, Q.49)**

(c) Discuss the candidate key, primary key, super key, composite key and alternate key. **(See Unit-II, Page 86, Q.58)**

Or

2. **(a)** Draw an E-R diagram for an Order Processing System, make suitable assumptions.

(b) What is the role DBA in Data Base Management System ? **(See Unit-I, Page 35, Q.44)**

(c) Explain the component modules of a DBMS and their interactions with the architecture. **(See Unit-I, Page 10, Q.10)**

3. **(a)** Write short note on the following –

(i) Network data Model **(See Unit-I, Page 19, Q.22)**

(ii) Hierarchical data model. **(See Unit-I, Page 18, Q.20)**

(b) Why are entity integrity and referential integrity important in database? Explain with suitable examples. **(See Unit-II, Page 47, Q.6)**

Or

Msinvent

(a) Define the following –
(i) Entity and attributes (See Unit-I, Page 39, Q.50)
(ii) Simple and composite attributes (See Unit-I, Page 42, Q.54)
(iii) Single value and Multivalue Attributes (See Unit-I, Page 42, Q.55)
(iv) Weak entity and Strong entity. (See Unit-II, Page 97, Q.67)

(b) Explain the advantages of Relational model and also explain the Data control facilities in Relational Model. (See Unit-I, Page 30, Q.34)

(a) Consider the following Relational Database
Employee (EMP_Name, Street, City)
Works (EMP_Name, Company_Name, Salary)
Company (Company_Name, City)
Manages (EMP_Name, Manager_Name)
For each of the following queries, give an expression in the Relational Algebra –
(i) Find the names of all employees who work for Satyam.
(ii) Find the names and cities of Residence who work for Satyam.
(iii) Find the names of all Employees who live in the same city as the company for which they work.
(iv) Find the names of all Employees who do not work for Satyam.
(v) Find the names of all employees who live in the same city and on the same street as do their managers.
(See Unit-II, Page 78, Q.45)

(b) Explain the basic relational algebra operation's with the symbol used and example for each. (See Unit-II, Page 52, Q.19)

Or

(a) Let the following relational schema be given :
Employee (SSN, name, age, dno)
Salary (SSN, salary)
Works_on (Projects#, SSN)
Project (Projects#, project_name, location)
For each of the following queries give an expression in tuple relational calculus :
(i) Display the names of projects at "Delhi".
(ii) Find the project_name of employee whose salary is greater than 10000
(iii) Retrieve the name and SSN of employees working on project#A 100.
(See Unit-II, Page 79, Q.47)

(b) Write short note on the following –
(i) Data Manipulation Language (DML) (See Unit-III, Page 135, Q.20)

(ii) Data Definition Language (DDL) (See Unit-III, Page 135, Q.20)
(iii) Transaction Control Statement (TCS) (See Unit-III, Page 135, Q.20)
(iv) Data Control Language (DCL) (See Unit-III, Page 135, Q.20)
(v) Data Administration Statements (DAS).

7. (a) What is Normalization ? Discuss various normal forms with the help of examples. (See Unit-IV, Page 215, Q.55)

(b) Consider the relation r(A, B, C, D, E) and the set $F = \{AB \rightarrow CE, E \rightarrow AB, C \rightarrow D\}$, What is the highest normal form of this relation. (See Unit-IV, Page 191, Q.24)

Or

8. (a) Define the following terms –
(i) Functional Dependency. (See Unit-IV, Page 179, Q.1)
(ii) Lossless Decomposition. (See Unit-IV, Page 198, Q.37)
(iii) Dependency Preservation. (See Unit-IV, Page 197, Q.36)
(iv) Third Normal Form. (See Unit-IV, Page 192, Q.25)

(b) Given the relation r (X, Y, W, Z, P, Q) and the set $F = \{XY \rightarrow W, XW \rightarrow P, PQ \rightarrow Z, XY \rightarrow Q\}$. Consider the decomposition $R_1(Z,P,Q), R_2(X,Y,Z,P,Q)$. Is this decomposition lossless or lossy ? Use the Lossless-join algorithm.
(See Unit-IV, Page 203, Q.41)

9. (a) Describe the three most common concurrent transaction execution problems. Explain how concurrency control can be used to avoid those problems. (See Unit-V, Page 233, Q.22)

(b) What do you mean by time stamping protocol for concurrency control ? Discuss multi-version scheme of concurrency control also.
(See Unit-V, Page 240, Q.35)

Or

10. (a) Explain 2-phase commitment protocol and the behavior of this protocol during lost messages and site failures.
(See Unit-V, Page 242, Q.36)

(b) List and discuss some of the factors that influenced the evaluation of the DDBMS. (See Unit-V, Page 268, Q.74)

MSInventer

RGPV

B.E. (Fourth Semester) EXAMINATION, June 2014
DATABASE MANAGEMENT SYSTEM
(IT–403)

: (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each question are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

(a) What is meant by instance of the database schema. 2

(See Unit-I, Page 31, Q.38)

(b) Compare specialization and generalization. (See Unit-II, Page 100, Q.72) 2

(c) Explain how the ER diagram is converted into tables. 3

(d) What is meant by mapping cardinalities ? For a binary relationship set what are the possible mapping cardinalities ? Explain with diagrams. (See Unit-II, Page 82, Q.52) 7

Or

With the help of block diagram explain the architecture of a DBMS (See Unit-I, Page 10, Q.10) 7

(a) Write characteristics of relational model. (See Unit-I, Page 23, Q.26) 2

(b) Specify CODD's norms. (See Unit-III, Page 131, Q.16) 2

(c) What is integrity constraint ? Explain the concept of referential integrity. (See Unit-II, Page 49, Q.8) 3

(d) Compare hierarchical, network and relational data model. Also discuss their various advantages and disadvantages. (See Unit-I, Page 30, Q.35)

Or

Discuss different update operations that can be performed on relations. (See Unit-II, Page 87, Q.61) 7

(a) Differentiate between relational algebra and relational calculus. 2

(See Unit-II, Page 73, Q.39)

(b) What are views ? Why they can't be used for updates ? 2

(See Unit-III, Page 127, Q.11)

(c) What is union compatibility ? What are the relational algebra operators that require the relations on which they are applied be union compatible ? (See Unit-II, Page 59, Q.23) 3

(d) Explain the following relational algebra operations – 7

(i) Natural join (ii) Assignment

(iii) Generalized projection (iv) Set intersection

(See Unit-II, Page 69, Q.35)

Or

For the given database, write SQL queries :

employee (eid, employee - name, street, city)

works (eid, cid, salary)

company (cid, company-name, city)

manager (eid, manager-name)

(i) Find the names, street and city of all employees who work for "ABC" and earn more than Rs. 20,000.

(ii) Find the names of all employees having "i" as the second letter in their names.

(iii) Display the annual salary of all employees. 7

(See Unit-III, Page 160, Prob.13)

4. (a) Specify the need of normalization. (See Unit-IV, Page 186, Q.15) 2

(b) State Armstrong's axioms of functional dependency. 2

(See Unit-IV, Page 181, Q.3)

(c) Consider relation R(PQRSTU) with following dependencies.

$P \rightarrow Q, ST \rightarrow PR, S \rightarrow U$

State R is in which normal form ? Decompose it to BCNF. Show step by procedure. 3

(d) Define normalization ? What is the importance of normalization in database design ? Explain 1 NF, 3 NF and BCNF with example. 7

(See Unit-IV, Page 215, Q.55)

Or

What is decomposition ? What are the desirable properties of decomposition ? (See Unit-IV, Page 196, Q.34) 7

5. (a) Explain different states of a transaction. (See Unit-V, Page 220, Q.4) 2

(b) Explain two phase locking protocol with example. 2

(See Unit-V, Page 234, Q.25)

(c) What is serializable schedule ? Describe with suitable example the types of serializable schedules. (See Unit-V, Page 227, Q.16) 3

(d) Explain data fragmentation, replication and allocation techniques for distributed database design. (See Unit-V, Page 264, Q.69) 7

MTU Universe

Or

Explain how the concept of object identify in the object oriented model differs from the concept of tuple equality in the relational model. 7

(See Unit-V, Page 274, Q.83)

RGPV

B.E. (Fourth Semester) EXAMINATION, Dec. 2014
DATABASE MANAGEMENT SYSTEM
(IT–403)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each question are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) Define the two level of data independence. (See Unit-I, Page 34, Q.42)

(b) What are the ACID properties ? (See Unit-V, Page 218, Q.2)

(c) List out the various relational algebra operator. (See Unit-II, Page 52, Q.18)

(d) Describe the component of entity relationship diagram with suitable examples. (See Unit-II, Page 93, Q.64)

Or

Describe the features of embedded SQL and dynamic SQL. Give suitable examples. (See Unit-III, Page 136, Q.21)

Unit-II

2. (a) What are the three kinds of intent locks ? (See Unit-V, Page 234, Q.24)

(b) What do you mean by weak entity set ? (See Unit-II, Page 97, Q.67)

(c) Write a note on functional dependencies. (See Unit-IV, Page 179, Q.1)

(d) Draw E-R diagram for a small telecom marketing company database, assuming your own data requirements. (See Unit-II, Page 114, Prob.8)

Or

Consider the universal relation –
R = {A, B, C, D, E, F, G, H, I, J} and the set of functional dependencies
F = {(A, B)} → {C}, {A} → {D, E}
{B} → {F}, {F} → {G, H},
{D} → {I, J} what is the key for R ?
Decompose R into 2 NF, then 3 NF relations.

(See Unit-IV, Page 194, Q.30)

Unit-III

3. (a) Consider the following relation –
R(A, B, C, D, E)
The primary key of the relation is AB.
The following functional dependencies hold :
A → C B → D AB → E
is the above relation in second normal form ?

(b) With an example explain what a derived attribute is ? (See Unit-I, Page 42, Q.56)

(c) List the commonly used concurrency control techniques. (See Unit-V, Page 236, Q.27)

(d) Define a functional dependency list and discuss the six interference rules for functional dependencies, give relevant example. (See Unit-IV, Page 181, Q.4)

Or

Discuss two phase locking protocol with suitable example. (See Unit-V, Page 234, Q.25)

Unit-IV

4. (a) Give the usage of rename operation with an example. (See Unit-II, Page 59, Q.20)

(b) Write down any two major responsibilities of a database administer. (See Unit-I, Page 36, Q.45)

(c) Define irreducible sets of dependencies. (See Unit-IV, Page 184, Q.12)

(d) Write short notes on the following –
(i) Mandatory access control (See Unit-V, Page 247, Q.44)
(ii) Missing information (See Unit-III, Page 127, Q.11)

Or

Discuss join dependencies on fifth normal form and explain why 5 NF ? (See Unit-IV, Page 213, Q.52)

Unit-V

5. (a) What is data tuning ? (See Unit-V, Page 229, Q.19)

(b) What are two pitfalls (problems) of lock-based protocols ? (See Unit-V, Page 237, Q.30)

(c) Distinguish between sparse index and dense index.

(d) Define a transaction with a neat sketch discuss the states a transaction can be in then discuss the following with suitable examples –
(i) A read only transaction
(ii) A read write transaction
(iii) An aborted transaction.

(See Unit-V, Page 221, Q.6)

MsInventer

Or

What is hashing ? Explain the distinction between closed and open hashing. Discuss the relative merits of each technique in database application.

RGPV

B.E. (Fourth Semester) EXAMINATION, June 2015
DATABASE MANAGEMENT SYSTEM
(IT–403)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each questions are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) What are the functions of a DBA ? (See Unit-I, Page 36, Q.45)
(b) List four significant differences between a file processing system and a DBMS. (See Unit-I, Page 7, Q.7)
(c) Define the two levels of data independence. (See Unit-I, Page 34, Q.42)
(d) Draw an E-R diagram corresponding to customers and loans. (See Unit-II, Page 97, Q.65)

Or

What are the different data models present ? Explain in detail. (See Unit-I, Page 17, Q.18)

Unit-II

2. (a) What are the four broad categories of constraints ? (See Unit-II, Page 46, Q.5)
(b) Discuss the basic concepts of network data model. (See Unit-I, Page 19, Q.22)
(c) Explain relational schema, intension and extension. (See Unit-II, Page 46, Q.4)
(d) What are the different types of integrity constraints ? Explain by giving suitable example. (See Unit-II, Page 47, Q.6)

Or

Explain various types of keys. With example. (See Unit-II, Page 84, Q.55)

Unit-III

3. (a) List the SQL statements used for transaction control. (See Unit-III, Page 135, Q.19)
(b) Write short note on relational calculus. (See Unit-II, Page 72, Q.38)
(c) Differentiate between SQL and QBE ? (See Unit-III, Page 141, Q.26)
(d) Consider the relational database given below –
lives (person_name, street, city)
works (person_name, company_name, salary)
located in (company_name, city)
manager (person_name, manager_name)
Give an expression in SQL for each of the queries below –
(i) Find the company with the smallest payroll
(ii) Find the company with the most employees
(iii) Give all managers a 10% raise. (See Unit-III, Page 149, Prob.3)

Or

What is union compatibility ? What are the relational algebra operators that require the relations on which they are applied be union compatible? (See Unit-II, Page 59, Q.23)

Unit-IV

4. (a) What do you mean by normalization ? (See Unit-IV, Page 186, Q.15)
(b) Consider the following relation –
R(A, B, C, D, E)
The primary key of the relation is AB. The following functional dependencies hold –
$A \rightarrow C$
$B \rightarrow D$
$AB \rightarrow E$
Is the above relation in second normal form ?
(c) Give an example of a relation schema R and a set of dependencies such that R is in BCNF, but not in 4 NF. (See Unit-IV, Page 212, Q.51)
(d) Give a set of functional dependencies for the relation schema R(A, B, C, D, E) with primary key AB under which R is in 2 NF but not in 3NF.

Or

Describe about the multi-valued dependencies and fourth normal form with suitable example. (See Unit-IV, Page 212, Q.50)

Unit-V

5. (a) List the two commonly used concurrency control techniques ? (See Unit-V, Page 236, Q.27)

MSUniversity

(b) What do you mean by a transaction ? (See Unit-V, Page 217, Q.1)

(c) What is database integrity ? Explain integrity constraints. (See Unit-V, Page 255, Q.54)

(d) What is distributed database system ? How is it differ from the centralized database system ? (See Unit-V, Page 258, Q.56)

Or

Give basic concepts of object-oriented database system and design. (See Unit-V, Page 271, Q.78)

RGPV

B.E. (Fourth Semester) EXAMINATION, Dec. 2015
DATABASE MANAGEMENT SYSTEM
(IT–403)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each question are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

1. (a) List any eight applications of DBMS. (See Unit-I, Page 6, Q.5)

(b) What are the advantages of using a DBMS ? (See Unit-I, Page 8, Q.8)

(c) Define instance and schema. (See Unit-I, Page 31, Q.39)

(d) What are the features and objectives of DBMS ? (See Unit-I, Page 3, Q.2)

Or

What are the responsibilities of a DBA ? If we assume that the DBA is never interested in running his or her own queries, does the DBA still need to understand query optimization ? Why ? (See Unit-I, Page 39, Q.47)

2. (a) Distinctions among the terms primary key, candidate key. (See Unit-II, Page 86, Q.57)

(b) What is a data model ? (See Unit-I, Page 16, Q.17)

(c) Explain the following terms briefly: attribute, domain, entity-relationship. (See Unit-I, Page 43, Q.60)

(d) What is E-R model ? What are the various symbols used to draw E-R diagram ? Draw E-R diagram of the educational organizations. (See Unit-II, Page 108, Prob.3)

Or

Explain various data models ? (See Unit-I, Page 17, Q.18)

3. (a) Define query language. (See Unit-III, Page 119, Q.1)

(b) What is a view in SQL ? (See Unit-III, Page 127, Q.12)

(c) Differentiate between inner join and outer join. (See Unit-II, Page 62, Q.27)

(d) Consider the following relations –

Student (snum : integer, sname : string, major string, level: string, age: integer)

Class (name : string, meetsat : string, room : string, fid : integer)

Enrolled (snum : integer, cname : string)

Faculty (fid : integer, fname : string, deptid : integer)

The meaning of these relations is straightforward : for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL.

(i) Find the names of all classes that either meet in room R128 or have five or more students enrolled.

(ii) Find the names of all students who are enrolled in two classes that meet at the same time. (See Unit-III, Page 160, Prob.14)

Or

Define the following term in context of SQL.

(i) Views (ii) Join (ii) In (See Unit-III, Page 130, Q.15)

4. (a) What are the uses of functional dependencies ? (See Unit-IV, Page 180, Q.2)

(b) What is first normal form ? (See Unit-IV, Page 187, Q.18)

(c) Define canonical cover. (See Unit-IV, Page 183, Q.10)

(d) Explain BCNF and 3NF and 5NF with suitable example. (See Unit-IV, Page 215, Q.53)

Or

Define multivalued dependency. What do you understand by trivial multivalued dependency ? (See Unit-IV, Page 211, Q.48)

5. (a) What does database security refer to ? (See Unit-V, Page 246, Q.40)

(b) What are the properties of transaction ? (See Unit-V, Page 218, Q.2)

(c) What is heterogeneous distributed database ? (See Unit-V, Page 262, Q.67)

MSAnswer

(d) Explain serializability. Give an example of a strict schedule that is not serializable. (See Unit-V, Page 224, Q.13)

Or

Discuss the concept of object oriented data base system. (See Unit-V, Page 270, Q.77)

RGPV

B.E. (Fourth Semester) EXAMINATION, June 2016
DATABASE MANAGEMENT SYSTEM
(IT–403)

ote : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each question are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

(a) Write any three functions of DBA. (See Unit-I, Page 36, Q.45)

(b) Differentiate between single valued and multivalued. (See Unit-I, Page 42, Q.55)

(c) Write any five advantages of D.B.M.S. (See Unit-I, Page 8, Q.8)

(d) What is data independence and why it is important ? (See Unit-I, Page 32, Q.40)

Or

Draw an E-R diagram of university by determining entities of interest and the relationships that exist between these entities. (See Unit-II, Page 108, Prob.4)

(a) Explain super key with suitable example. (See Unit-II, Page 86, Q.56)

(b) Write any two advantages of network data model. (See Unit-I, Page 30, Q.33)

(c) How foreign key is useful to maintain referential integrity ? (See Unit-II, Page 89, Q.62)

(d) What are different types of data models ? Explain relational data model with the help of some suitable example and compare with network data model. (See Unit-I, Page 27, Q.31)

Or

Explain following term
(i) Domains (ii) Tuples (iii) Schemas (See Unit-II, Page 51, Q.16)

3. (a) Discuss where Natural Join operation is used in relational algebra. (See Unit-II, Page 61, Q.25)

(b) Explain following command with an example
(i) Insert (ii) Group by (See Unit-III, Page 126, Q.10)

(c) What is NULL ? Give an example to illustrate testing for NULL in SQL. (See Unit-III, Page 132, Q.17)

(d) How the modification of database can be done in QUEL ? Explain. (See Unit-III, Page 137, Q.23)

Or

Consider the following relations with keys underlined
Street (name, location, city)
House (number, street_name)
Lives (name, house_number)
For the above relations answer the following queries in SQL.
(i) Get the house numbers street wise.
(ii) Get the numbers of houses which are not occupied. (See Unit-III, Page 147, Prob.1)

4. (a) Write the purpose of Normalization in DBMS. (See Unit-IV, Page 187, Q.17)

(b) Explain Trivial and non-Trivial dependencies. (See Unit-IV, Page 181, Q.5)

(c) What is partial dependency ? With which normal form is it associated ? (See Unit-IV, Page 182, Q.7)

(d) Describe NULL value and dangling tuple problems. (See Unit-III, Page 132, Q.18)

Or

What are the inference rules for functional dependencies ? (See Unit-IV, Page 181, Q.3)

5. (a) Write any two main objectives of distributed system. (See Unit-V, Page 260, Q.60)

(b) Write the basic properties of a transaction. (See Unit-V, Page 218, Q.2)

(c) Explain integrity concept with example. (See Unit-V, Page 255, Q.54)

(d) What is distributed system ? How is it differ from the centralized database system ? Give the uses of distributed system. (See Unit-V, Page 260, Q.62)

Or

MSInventer

Write short notes :
(i) Recovery (See Unit-V, Page 247, Q.45)
(ii) Concurrent operation (See Unit-V, Page 221, Q.7)

RGPV B.E. (Fourth Semester) EXAMINATION, Dec. 2016
DATABASE MANAGEMENT SYSTEM
(IT–403)

Note : (i) Answer any five questions.
(ii) All questions carry equal marks.

1. (a) Discuss the various components of database system. 7
(See Unit-I, Page 10, Q.10)

(b) What are the different levels of data abstraction ? Explain each of them. (See Unit-I, Page 16, Q.16) 7

2. (a) Draw an E-R diagram of university by determining entities of interest and the relationship that exists between there entities. 7
(See Unit-II, Page 108, Prob.4)

(b) What are data models ? Discuss briefly any one data model. 7
(See Unit-I, Page 19, Q.21)

3. (a) What are the different types of database end users ? Discuss the main activities of each. (See Unit-I, Page 12, Q.11) 7

(b) Explain about set operation in relational algebra. 7
(See Unit-III, Page 123, Q.7)

4. (a) What is referential integrity ? How it is implemented using SQL. 7
(See Unit-II, Page 48, Q.7)

(b) Draw an E-R diagram for a hospital management system. Assume suitable entity and relations. (See Unit-II, Page 110, Prob.6) 7

5. (a) Explain join, natural join, outer join, full outer join, left outer join and right outer join with example. (See Unit-II, Page 69, Q.36) 7

(b) What is BCNF ? Give an example of a relation schema R and set of dependencies such that R is in BCNF, but not in 3NF. 7
(See Unit-IV, Page 209, Q.46)

6. (a) What are the inference rules for multivalued dependencies ? Explain. 7
(See Unit-IV, Page 181, Q.3)

(b) Discuss the important design goals for relational database, and explain why each is desirable. (See Unit-IV, Page 206, Q.44) 7

7. (a) Discuss the relative advantages of centralized and distributed database. (See Unit-V, Page 261, Q.64) 7

(b) What are object oriented databases ? What are its advantages and disadvantages ? (See Unit-V, Page 272, Q.80) 7

8. Write short notes –
(a) Security and integrity constraints (See Unit-V, Page 272, Q.55)
(b) Functional dependency (See Unit-IV, Page 179, Q.1)
(c) Concurrent operations (See Unit-V, Page 221, Q.7)
(d) Problems with dangling tuple. (See Unit-III, Page 132, Q.18)

RGPV B.E. (Fourth Semester) EXAMINATION, June 2017
Choice Based Credit System (CBCS)
DATABASE MANAGEMENT SYSTEM
(IT–225)

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Explain the responsibilities of the DBA and the database designers ?
(See Unit-I, Page 39, Q.49)

(b) Define the following terms –
(i) Entity
(ii) Attribute
(iii) Multivalued attribute.
(See Unit-I, Page 42, Q.57)

2. (a) What do you mean by database management system ? Explain the various advantages of using a database management system ?
(See Unit-I, Page 9, Q.9)

(b) Discuss the entity integrity and referential integrity constraints. Why is each considered important ? (See Unit-II, Page 47, Q.6)

3. (a) Draw an E-R diagram for the hospital management system. Assume your own entities (minimum of 5 entities), attributes and relations. Explain in detail. (See Unit-II, Page 109, Prob.5)

(b) Discuss the different relational algebra operations.
(See Unit-II, Page 63, Q.29)

4. (a) With an example explain clearly JOIN and UNION operations in relational algebra. Bring out the difference between natural JOIN and OUTER JOIN. (See Unit-II, Page 62, Q.26)

(b) Consider the following relations for a database that keeps track of business trips of sales persons in a sales office : Salesperson

MeriVenter

(Salespersonid, Name, Start-year, Dept-no.) Trip (Salespersonid, from, to, Departure-date, return-date, trip-id).
Expense (trip-id, Account No., Amount)
Specify the foreign keys for the above schema. Then specify the following queries in
Relational algebra –

(i) Give the details (all attributes of trip relation) for trip that exceeded 10,000/- in expenses.
(ii) Print the 'Salespersonid' and 'Name' of the salespersons who took trips to 'delhi'.
(iii) Print the total trip expenses incurred by the salesman with Salespersonid = '504'.

(See Unit-II, Page 79, Q.46)

(a) Consider the universal relation R = [A, B, C, D, E, F, G, H, I, J] and the set of functional dependencies –
F = {{A, B} –> {C}, {A} –> {D, E}, {B} –> {F}, {F} –> {G, H}, {D} –> {I, J}}
What is the key for R ? Decompose R into 2NF, then 3NF relations.

(See Unit-IV, Page 194, Q.30)

(b) Consider the following schema for a company database employee (Name, SSN, Address, Sex, Salary, Dno)
Department (Dname, Dnumber, MGRSSN, MGRSTART Date)
Dept-Locations (Dnumber, Dlocations)
Project (Pname, Pnumber, Plocations, Dnum)
Works-On (ESSN, PNo, Hours)
Dependent (ESSN, Dependent-name, Sex, Bdate, Relationship)
Give the queries in SQL

(i) Retrieve the names and address of employees who work for "Research" department.
(ii) List all the project names on which employee "Smith" is working.
(iii) Retrieve all employees who either work in department 4 and make over 25000 per year or work in department 5 and make over 30,000.
(iv) Retrieve the SSN of all employees who either work in department 5 or directly supervise an employee who works in department number.

(See Unit-III, Page 158, Prob.11)

(a) Define BCNF. How does it differ from 3NF ? What is it considered a stronger form of 3NF ? Explain with neat diagram.

(See Unit-IV, Page 204, Q.42)

(b) Explain each of the following with example –
(i) First normal form
(ii) Second normal form
(iii) Third normal form

(See Unit-IV, Page 194, Q.29)

7. (a) What is serialisability ? How can serializability be ensured ? Do you need to restrict.

(See Unit-V, Page 238, Q.32)

(b) Give below are two sets of FDs for a relation R(A, B, C, D, E). Are they equivalent.
(i) A –> B, AB –> C.D –> AC, D –> E
(ii) A –> BC, D –> AE

(See Unit-IV, Page 184, Q.13)

8. (a) Explain how strict 2-phase locking is implemented. Show them with an example.

(See Unit-V, Page 236, Q.29)

(b) Concurrent execution of transaction to ensure serializability ? Justify your answer. Give an example of transactions and how you can force serializability in those transactions.

(See Unit-V, Page 239, Q.34)

RGPV

IT-225 (CBCS)
B.E. (Fourth Semester) EXAMINATION, Dec. 2017
Choice Based Credit System (CBCS)
DATABASE MANAGEMENT SYSTEM

Note : (i) Total number of questions 08.
(ii) Attempt any five questions (including all parts).
(iii) Assume missing data, if any, suitably.

1. (a) State comparison of RDBMS, OODBMS and ORDBMS.

(See Unit-V, Page 272, Q.81)

(b) With a neat sketch discuss the three schema architecture of a DBMS.

(See Unit-I, Page 15, Q.14)

2. (a) Explain the following –
(i) Aggregation
(ii) Strong and weak entity set
(iii) Generalization/specialization

(See Unit-II, Page 104, Q.76)

(b) For the following problem definition –
The musical company wants to store information about the musicians who perform on its albums. Each musician has a musician Id, a

MSinVenter

name, an address, and a phone number. Some musicians may have the same address and some of them may have more than one phone number. Each musician may play several instruments and an instrument and may be played by several musicians. Each instrument has name and a musical key. The album recorded has a title, a copyright date, a format and an album identifier. Each album has a number of songs where a song has a title and an author. Each song may be performed by several musicians and a musician may perform a number of songs. One of the musicians of the song acts as a producer. A producer may produce several albums.

(i) Draw an E-R diagram.

(ii) Transform the E-R diagram to a Relational Schema.

(See Unit-II, Page 117, Prob.10)

3. (a) Describe the steps used to perform JOINS in parallel database.

(See Unit-III, Page 172, Q.46)

(b) Define domain integrity constraint, entity integrity constraint and referential integrity constraint. (See Unit-II, Page 49, Q.11)

4. (a) Explain in detail about heuristic approach of query optimization.

(See Unit-III, Page 164, Q.36)

(b) Explain multi-valued dependencies and also describe 4NF with appropriate example. (See Unit-IV, Page 212, Q.50)

5. (a) Discuss the various aggregate functions in SQL with an example for each. (See Unit-III, Page 120, Q.4)

(b) What is Deadlock ? Discuss the different ways of preventing deadlock and detecting deadlock. (See Unit-V, Page 243, Q.37)

6. (a) Explain concurrency control and recovery in distributed database management system. (See Unit-V, Page 249, Q.47)

(b) Suppose that we decompose the schema R = {P, Q, R, S, T} into R_1 = {P, Q, R} and R_2 = {P, S, T}. Verify whether this decomposition is Lossless or Lossy if the functional dependencies P → QR, RS → T, Q→ S and T → P holds. Also verify whether the decomposition is dependency preserving. (See Unit-IV, Page 201, Q.39)

7. (a) Consider the relation schema R(P, Q, R, S) and the functional dependencies PQ → R, R → S and S → P holds on R. List all the keys of relation R. (See Unit-IV, Page 216, Q.56)

(b) Explain the concept of conflict serializability. Define conflicting instructions and non-conflicting instructions.

(See Unit-V, Page 225, Q.15)

8. Write short notes on any three of the following –

(a) BCNF (See Unit-IV, Page 204, Q.42)

(b) DB2 (See Unit-III, Page 177, Q.51)

(c) Schemas and instances (See Unit-I, Page 31, Q.39)

(d) Web based data model (See Unit-I, Page 21, Q.24)

(e) Locking techniques (See Unit-V, Page 236, Q.28)

RGPV

IT-4002 (CBGS)

B.E. (Fourth Semester) EXAMINATION, May 2018

Choice Based Grading System (CBGS)

DATABASE MANAGEMENT SYSTEM

Note : (i) Attempt any five questions.

(ii) All questions carry equal marks.

1. (a) What is the difference between database user and database administrator ? Explain various functions of database administrator. 7

(See Unit-I, Page 38, Q.46)

(b) Draw and discuss three levels of Database Management System. 7

(See Unit-I, Page 15, Q.14)

2. (a) What is relational algebra ? Enlist and explain the fundamental operations of relational algebra. (See Unit-II, Page 59, Q.21) 7

(b) Explain disadvantages of conventional file based system compared to database management system. (See Unit-I, Page 7, Q.7) 7

3. (a) Define E-R diagram. Draw an E-R diagram for library management system. Assume relevant entities and attributes for given system. 7

(See Unit-II, Page 80, Q.48)

(b) What is aggregate function ? Explain any five aggregate functions with example. (See Unit-III, Page 120, Q.4) 7

4. (a) What are integrity constraints ? Explain various types of integrity constraints with suitable example. (See Unit-II, Page 47, Q.6) 7

(b) Discuss various types of joins with suitable example. 7

(See Unit-II, Page 69, Q.36)

5. (a) Write SQL statement for the following – 7

Student (Enrno, Name, CourseID, EmailID, CellNo.)

Course (CourseID, Course-nm, duration)

(i) Add a column city in student table

(ii) Find out list of students who have enrolled in "Computer" course.

(iii) List name of all course with their duration

(iv) List name of all students starting with "a".

(v) List EmailID and cellno of all mechanical engineering students.

(See Unit-III, Page 161, Prob.15)

(b) Give a brief note one query processing and query optimization. 7

(See Unit-III, Page 164, Q.34)

6. (a) What is normalization ? Why is it required ? Explain 1 NF, 2 NF and 3 NF with suitable examle. (See Unit-IV, Page 193, Q.28)

(b) What is concurrency control ? What are the recovery techniques to overcome from concurrency ? (See Unit-V, Page 249, Q.47)

7. (a) Suppose that we decompose the scheme R = (A, B, C, D, E) into (A, B, C) and (A, D, E). Show that this decomposition is a lossless join decomposition if the following function dependency hold -
$A \rightarrow BC$, $CD \rightarrow E$, $B \rightarrow D$ (See Unit-IV, Page 202, Q.40)

(b) Define transaction. Discuss ACID proportion of a transaction with relevant example. (See Unit-V, Page 218, Q.3)

8. Write short notes –

(a) Distributed databases (See Unit-V, Page 259, Q.58)

(b) Two phase locking protocol (See Unit-V, Page 234, Q.25)

(c) Log based recovery (See Unit-V, Page 248, Q.46)

Msinventer